COVID-19 NPIS

Release 1.0.0

Priesemann group

May 19, 2021

CONTENTS

Covid-19

Hierachical Bayesian Model

CHAPTER ONE

INSTALLATION

We use some functions of our toolbox for inference and forecast of the spread of the Coronavirus. We supply this toolbox via a github submodule, which can to be initialized while cloning the repository. Alternatively our toolbox can be installed with pip.

```
git clone --recurse-submodules git@github.com:Priesemann-Group/covid19_npis_europe.git
```

Next install all required python libaries. Tensorflow isn't put in the requirements.txt to allow the installation of another distribution (tensorflow-gpu for instance)

```
pip install tensorflow==2.4.1
pip install -r requirements.txt
```

Now we should be setup and ready to run our code or look into the source code.

CHAPTER

TWO

RUNNING THE SIMULATION(S)

If you want to run our code there is a work flow you can follow.

1. **Create or generate a dataset.** We supply multiple scripts to generate different datasets. All of them can be found in the *scripts/data_generators/* folder. All of them create the data files inside the *data* folder. You can run them easily with

   ```
   python script.py
   ```

   Alternatively you create your own dataset, we wrote are short guide which can help you get your data inserted into our model see here.

2. **Load dataset** Before we can start to fit with our model we have to load our data files. There are multiple ways to do this but all of them rely on the `covid19_npis.ModelParams` class. Have a look into the constructors to see all possibilities.

   The easiest way is to load a complete data folder e.g. *data/Germany_bundesländer* (generated with the Germany_bundesländer.py script).

   ```
   modelParams = covid19_npis.ModelParams.from_folder("./data/Germany_
   ↪bundesländer/")
   ```

3. **Generate model with data** Now according to the dimensions of our data (i.e. time,number of countries. . .) we create our model. This is done by passing the *modelParams* to our pymc4 model instance.

   ```
   this_model = covid19_npis.model.main_model(modelParams)
   ```

4. **Sampling** Sampling is done with the `pymc4.sample()` function and our model from above. The sampling function generates an `arviz.InferenceData` object, which we can later use for the plotting or for other sample stats.

   ```
   # A typical sample function call
   begin_time = time.time()
   log.info("Start sampling")
   trace = pm.sample(
           this_model,
           burn_in=200,
           num_samples=100,
           num_chains=2,
           xla=True,
           step_size=0.01,
   )
   end_time = time.time()
   log.info("running time: {:.1f}s".format(end_time - begin_time))
   ```

(continues on next page)

(continued from previous page)

```
Best practise is to measure the time the sampling takes and to save the trace␣
↪after sampling.

# Save the trace
name, fpath = covid19_npis.utils.save_trace(
    trace, modelParams, fpath="./traces",
)
```

5. **Plotting** Todo

CHAPTER

THREE

UNDERSTANDING OUR MODEL

We supply our model which we used in our publication (wip). If you want to know how it works in detail have a look into our Methods section in the publication and the documentation here. You can also use our functions to create your own model but that could take some effort.

We suggest you start with the *covid19_npis.model.main_model* and work your way threw from top to bottom. It is always helpful to have the tensorflow documentation. opened. We use `tf.einsum` so you should have a look at Einstein notation if you don't know it by heart yet.

CHAPTER

FOUR

MODEL

4.1 Main model

`covid19_npis.model.`**`main_model`**`(`*modelParams*`)`

Todo: Create Docstring for this function.

4.2 Disease spread

`covid19_npis.model.disease_spread.`**`InfectionModel`**`(`*N*, *E_0_t*, *R_t*, *C*, *gen_kernel*`)`

This function combines a variety of different steps:

1. Converts the given E_0 values to an exponential distributed initial E_{0_t} with an length of l this can be seen in `_construct_E_0_t()`.
2. Calculates R_{eff} for each time step using the given contact matrix C:

$$R_{diag} = \text{diag}(\sqrt{R})$$
$$R_{eff} = R_{diag} \cdot C \cdot R_{diag}$$

3. Calculates the $\tilde{I}$ arrays i.e. new infectious for each age group and country, with the efficient reproduction matrix R_{eff}, the susceptible pool S, the population size N and the generation interval $g(\tau)$. This is done recursive for every time step.

$$\tilde{I}(t) = \frac{S(t)}{N} \cdot R_{eff} \cdot \sum_{\tau=0}^{t} \tilde{I}(t-1-\tau) g(\tau)$$
$$S(t) = S(t-1) - \tilde{I}(t-1)$$

Parameters

- **`E_0`** – Initial number of infectious. batch_dims, country, age_group
- **`R_t`** – Reproduction number matrix. time, batch_dims, country, age_group
- **`N`** – Total population per country country, age_group

- **C** – inter-age-group Contact-Matrix (see 8) country, age_group, age_group
- **gen_kernel** – Normalized PDF of the generation interval batch_dims(?), l

Returns Sample from distribution of new, daily cases

`covid19_npis.model.disease_spread.`**`construct_generation_interval`**(*name='g', mu_k=120.0, mu_theta=0.04, theta_k=8.0, theta_theta=0.1, l=16*)

Generates the generation interval with two underlying gamma distributions for mu and theta

$$g(\tau) = Gamma(\tau; k = \frac{\mu_{D_{\text{gene}}}}{\theta_{D_{\text{gene}}}}, \theta = \theta_{D_{\text{gene}}})$$

whereby the underlying distribution are modeled as follows

$$\mu_{D_{\text{gene}}} \sim Gamma(k = 4.8/0.04, \theta = 0.04)$$
$$\theta_{D_{\text{gene}}} \sim Gamma(k = 0.8/0.1, \theta = 0.1)$$

Parameters

- **name** (*string*) – Name of the distribution for trace and debugging.
- **mu_k** (*number, optional*) – Concentration/k parameter for underlying gamma distribution of mu ($\mu_{D_{\text{gene}}}$). 120
- **mu_theta** (*number, optional*) – Scale/theta parameter for underlying gamma distribution of mu ($\mu_{D_{\text{gene}}}$). 0.04
- **theta_k** (*number, optional*) – Concentration/k parameter for underlying gamma distribution of theta ($\theta_{D_{\text{gene}}}$). 8
- **theta_theta** (*number, optional*) – Scale/theta parameter for underlying gamma distribution of theta ($\theta_{D_{\text{gene}}}$). 0.1
- **l** (*number, optional*) – Length of generation interval i.e t in the formula above 16

Returns Normalized generation interval pdf

`covid19_npis.model.disease_spread.`**`construct_E_0_t`**(*modelParams, len_gen_interv_kernel, R_t, mean_gen_interv, mean_test_delay=10*)

Generates a prior for E_0_t, based on the observed number of cases during the first 5 days. Currently it is implemented to take the first value of R_t, and multiply the inverse of R_t with first observed values until the begin of the simulation is reached. This is then used as a prior for a lognormal distribution which set the E_0_t.

Parameters

- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **len_gen_interv_kernel** (*number*) – . . . some description
- **R_t** (*tf.tensor*) – Time dependent reproduction number tensor $R(t)$. time, batch, country, age group
- **mean_gen_interv** (*countries*) – . . . some description

- **mean_test_delay** (*number, optional*) – . . . some description 10

Returns

E_0_t: some description time, batch, country, age_group

`covid19_npis.model.disease_spread.`**`construct_delay_kernel`**(*name, modelParams, loc, scale, length_kernel*)

Constructs delay d in hierarchical manner:

$$\begin{aligned}\mu_c^d &\sim \text{LogNormal}\left(\mu = 2.5, \sigma = 0.1\right) \quad \forall c\\ \sigma_c^d &\sim\\ d_c &= \text{PDF-Gamma}(\mu_c^d, \sigma_d)\end{aligned}$$

Parameters

- **name** – Name of the delay distribution
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **loc** – Location of the hierarchical Lognormal distribution for the mean of the delay.
- **scale** – Theta parameter for now#
- **length_kernel** – Length of the delay kernel in days.

Returns Generator for gamma probability density function. batch, country, kernel(time)

Todo: Think about sigma distribution and how to parameterize it. Also implement that.

4.3 Reproduction number

`covid19_npis.model.reproduction_number.`**`_fsigmoid`**(*t, l, d*)

Calculates and returns

$$\frac{1}{1 + e^{-4/l*(t-d)}}$$

Parameters

- **t** – Time, "variable"
- **l** – Length of the change point, determines scale
- **d** – Date of the change point, determines location

`covid19_npis.model.reproduction_number.`**`_create_distributions`**(*modelParams*)

Returns a dict of distributions for further processing/sampling with the following priors:

$$\begin{aligned}\alpha_i^\dagger &\sim \mathcal{N}\left(-1, 2\right) \quad \forall i,\\ \Delta\alpha_c^\dagger &\sim \mathcal{N}\left(0, \sigma_{\alpha,\text{country}}\right) \quad \forall c,\\ \Delta\alpha_a^\dagger &\sim \mathcal{N}\left(0, \sigma_{\alpha,\text{age}}\right) \quad \forall a,\\ \sigma_{\alpha,\text{country}} &\sim HalfNormal\left(0.1\right),\\ \sigma_{\alpha,\text{age}} &\sim HalfNormal\left(0.1\right)\end{aligned}$$

$$\begin{aligned}
l^\dagger_{\text{positive}} &\sim \mathcal{N}\left(3, 1\right), \\
l^\dagger_{\text{negative}} &\sim \mathcal{N}\left(5, 2\right), \\
\Delta l^\dagger_i &\sim \mathcal{N}\left(0, \sigma_{l, \text{interv.}}\right) \quad \forall i, \\
\sigma_{l, \text{interv.}} &\sim HalfNormal\left(1\right) \\
\Delta d_i &\sim \mathcal{N}\left(0, \sigma_{d, \text{interv.}}\right) \quad \forall i, \\
\Delta d_c &\sim \mathcal{N}\left(0, \sigma_{d, \text{country}}\right) \quad \forall c, \\
\sigma_{d, \text{interv.}} &\sim HalfNormal\left(0.3\right), \\
\sigma_{d, \text{country}} &\sim HalfNormal\left(0.3\right)
\end{aligned}$$

Parameters **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.

Returns interventions, distributions

covid19_npis.model.reproduction_number.**construct_R_t**(*name*, *modelParams*, *R_0*, *include_noise=True*)

Constructs the time dependent reproduction number $R(t)$ for every country and age group. There are a lot of things happening here be sure to check our paper for more indepth explanations!

We build the effectivity in an hierarchical manner in the unbounded space:

$$\begin{aligned}
\alpha_{i,c,a} &= \frac{1}{1+e^{-\alpha^\dagger_{i,c,a}}}, \\
\alpha^\dagger_{i,c,a} &= \alpha^\dagger_i + \Delta \alpha^\dagger_c + \Delta \alpha^\dagger_{a}
\end{aligned}$$

The length of the change point depends on the intervention and whether the strength is increasing or decreasing:

$$\begin{aligned}
l_{i, \text{sign}(\Delta \gamma)} &= \ln \left(1 + e^{l^\dagger_{i, \text{sign}(\Delta \gamma)}} \right), \\
l^\dagger_{i, \text{sign}(\Delta \gamma)} &= l^\dagger_{\text{sign}(\Delta \gamma)} + \Delta l^\dagger_i,
\end{aligned}$$

The date of the begin of the intervention is also allowed to vary slightly around the date $d^{\text{data}}_{i,c}$ given by the Oxford government response tracker:

$$d_{i,c,p} = d^{\text{data}}_{i,c,p} + \Delta d_i + \Delta d_c$$

And finally the time dependent reproduction number R^*_e:

$$\begin{aligned}
\gamma_{i,c,p}(t) &= \frac{1}{1+e^{-4/l_{i, \text{sign}(\Delta \gamma)} \cdot (t - d_{i,c,p})}} \cdot \Delta \gamma^{\text{data}}_{i,c,p} \\
\gamma_{i,c}(t) &= \sum_p \gamma_{i,c,p}(t) \\
R^*_e &= R^*_0 e^{-\sum_i^{N_i} \alpha_{i,c,a} \gamma_{i,c}(t)}
\end{aligned}$$

We also sometimes call the time dependent reproduction number R_t!

Parameters

- **name** (*str*) – Name of the distribution (gets added to trace).
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **R_0** (*tf.tensor*) – Initial reproduction number. Should be constructed using *construct_R_0()* or *construct_R_0_old()*. batch, country, age group

Returns Time dependent reproduction number tensor $R(t)$. time, batch, country, age group covid19_npis.model.reproduction_number.**construct_R_0**(*name*, *modelParams*, *loc*, *scale*, *hn_scale*)

Constructs R_0 in the following hierarchical manner:

$$\begin{aligned} R^*_{0,c} &= R^*_0 + \Delta R^*_{0,c}, \\ R^*_0 &\sim \mathcal{N}\left(2, 0.5\right) \\ \Delta R^*_{0,c} &\sim \mathcal{N}\left(0, \sigma_{R^*, \text{country}}\right) \quad \forall c, \\ \sigma_{R^*, \text{country}} &\sim HalfNormal\left(0.3\right) \end{aligned}$$

Parameters

- **name** (*str*) – Name of the distribution (gets added to trace).
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **loc** (*number*) – Location parameter of the R^*_0 Normal distribution.
- **scale** (*number*) – Scale paramter of the R^*_0 Normal distribution.
- **hn_scale** (*number*) – Scale parameter of the sigma_{R^*, text{country}} HaflNormal distribution.

Returns R_0 tensor batch, country, age_group

covid19_npis.model.reproduction_number.**construct_lambda_0**(*name*, *modelParams*, *loc*, *scale*, *hn_scale*)

Constructs lambda_0 in the following hierarchical manner:

$$\begin{aligned} \lambda^*_{0,c} &= \lambda^*_0 + \Delta \lambda^*_{0,c}, \\ \lambda^*_0 &\sim \mathcal{N}\left(0.4, 0.1\right) \\ \Delta \lambda^*_{0,c} &\sim \mathcal{N}\left(0, \sigma_{\lambda^*, \text{country}}\right) \quad \forall c, \\ \sigma_{\lambda^*, \text{country}} &\sim HalfNormal\left(0.05\right) \end{aligned}$$

Parameters

- **name** (*str*) – Name of the distribution (gets added to trace).
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **loc** (*number*) – Location parameter of the R^*_0 Normal distribution.
- **scale** (*number*) – Scale paramter of the R^*_0 Normal distribution.
- **hn_scale** (*number*) – Scale parameter of the sigma_{R^*, text{country}} HaflNormal distribution.

Returns R_0 tensor batch, country, age_group

covid19_npis.model.reproduction_number.**construct_R_0_old**(*name*, *modelParams*, *mean*, *beta*)

Old constructor of R_0 using a gamma distribution:

$$R_0 \sim Gamma\left(\mu=2.5, \beta=2.0\right)$$

Parameters

- **name** (*string*) – Name of the distribution for trace and debugging.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.

- **mean** – Mean μ of the gamma distribution.
- **beta** – Rate β of the gamma distribution.

Returns R_0 tensor batch, country, age_group

4.4 Number of tests

`covid19_npis.model.number_of_tests.`**`weekly_modulation`**(*name*, *modelParams*, *cases*)

Adds a weekly modulation of the number of new cases:

$$\text{cases\_modulated} = \text{cases} \cdot (1 - f(t)), \qquad \text{with}$$
$$f(t) = (1 - w) \cdot \left(1 - \left|\sin\left(\frac{\pi}{7}t - \frac{1}{2}\Phi_w\right)\right|\right)$$

The modulation is assumed to be the same for all age-groups within one country and determined by the "weight" and "offset" parameters. The weight follows a sigmoidal distribution with normal prior of "weight_cross". The "offset" follows a VonMises distribution centered around 0 (Mondays) and a wide SD (concentration parameter = 2).

Parameters

- **name** (*str or None,*) – The name of the cases to be modulated (gets added to trace).
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **cases** (*tf.tensor*) – The input array of daily new cases for countries and age groups

Returns **cases_modulated**

Return type tf.tensor

Todo:

- check prior parameters
- different modulations across: age, country?
- check: are (cumulative) case numbers same as in unmodulated case? need some kind of normalization?
- store and plot parameters at end

`covid19_npis.model.number_of_tests.`**`generate_testing`**(*name_total*, *name_positive*, *modelParams*, *new_E_t*)

High level function for generating/simulating testing behaviour.

Constructs B splines Delay cases

Parameters

- **name_total** (*str,*) – Name for the total tests performed
- **name_positive** (*str,*) – Name for the positive tests performed
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **new_E_t** (*tf.Tensor*) – New cases $E_{\text{age},a}$. batch, time, country, age_group

Returns $(n_{\Sigma,c,a}(t), n_{+,c,a}(t)$ Total and positive tests by age group and country (batch, time, country, age_group) x 2

Todo:

- Add more documenation for this function

covid19_npis.model.number_of_tests.**_calc_positive_tests**(*new_E_t_delayed*, *phi_plus*, *phi_age*)

$$n_{+,c,a}(t) = \tilde{E}_{\text{delayTest},c,a}(t) \cdot \phi_{+,c}(t) \cdot \phi_{\text{age},a},$$

Parameters

- **name** (*str*) – Name of the variable for the new positive cases $n_{+,c,a}(t)$ in the trace.
- **new_E_t_delayed** (*tf.Tensor*) – New cases with reporting delay $\tilde{E}_{\text{delayTest},c,a}(t)$. batch, time, country, age_group
- **phi_plus** (*tf.Tensor*) – Fraction of positive tests $\phi_{+,c}(t)$. batch, time, country
- **phi_age** (*tf.Tensor*) – Fraction of positive tests $\phi_{\text{age},a}$. batch, age_group

Returns $n_{+,c,a}(t)$ batch, time, country, age_group

covid19_npis.model.number_of_tests.**_calc_total_number_of_tests_performed**(*new_E_t_delayed*, *phi_tests_reported*, *phi_plus*, *eta*, *xi*)

$$\begin{aligned} n_{\Sigma,c,a}(t) = \phi_{\text{tests reported},c} \\ \cdot (\tilde{E}_{\text{delayTest},c,a}(t) \cdot \phi_{+,c}(t) \\ + \tilde{E}_{\text{delayTest},c,a}(t) \cdot \phi_{+,c}(t) \cdot \eta_{\text{traced},c}(t) \\ + \xi_c(t)) \end{aligned}$$

Parameters

- **name** (*str*) – Name of the variable for the total number of tests performed $n_{\Sigma,c,a}(t)$ in the trace.
- **new_E_t_delayed** (*tf.Tensor*) – New cases with reporting delay $\tilde{E}_{\text{delayTest},c,a}(t)$. batch, time, country, age_group
- **phi_tests_reported** (*tf.Tensor*) – Difference in fraction for different countries $\phi_{\text{tests reported},c}$ batch, country
- **phi_plus** (*tf.Tensor*) – Fraction of positive tests $\phi_{+,c}(t)$. batch, time, country
- **eta** (*tf.Tensor*) – Number of traced persons per case with subsequent negative test per case $\eta_{\text{traced},c}(t)$. batch, time, country
- **xi** (*tf.Tensor*) – Base rate of testing per day that leads to negative tests $\xi_c(t)$. batch, time, country

Returns $n_{\Sigma,c,a}(t)$ batch, time, country, age_group `covid19_npis.model.number_of_tests._construct_phi_tests_reported`(*name*, *modelParams*, *mu_loc=1.0*, *mu_scale=1.0*, *sigma_scale=1.0*)

Construct the different of the fraction of tests for each country in the following hierarchical manner:

$$\begin{aligned}\phi_{\text{tests reported},c} &= \frac{e^{\phi^\dagger_{\text{tests reported},c}}}{e^{\phi^\dagger_{\text{tests reported},c}}+1},\\ \phi^\dagger_{\text{tests reported},c} &\sim \mathcal{N}(\mu_{\phi^\dagger_{\text{tests reported}}},\sigma_{\phi^\dagger_{\text{tests reported}}}),\\ \mu_{\phi^\dagger_{\text{tests reported}}} &\sim \mathcal{N}(1,1),\\ \sigma_{\phi^\dagger_{\text{tests reported}}} &\sim HalfCauchy(1).\end{aligned}$$

Parameters

- **name** (*str*) – Name of the variable $\phi_{\text{tests reported},c}$. Will also appear in the trace with this name.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **mu_loc** (*optional*) – Location parameter for the Normal distribution $\mu_{\phi^\dagger_{\text{tests reported}}}$. 1.0
- **mu_scale** (*optional*) – Scale parameter for the Normal distribution $\mu_{\phi^\dagger_{\text{tests reported}}}$. 1.0
- **sigma_scale** (*optional*) – Scale parameter for the $\sigma_{\phi^\dagger_{\text{tests reported}}}$ HalfCauchy distribution. 1.0

Returns $\phi_{\text{tests reported},c}$ batch, country

`covid19_npis.model.number_of_tests._construct_phi_age`(*name*, *modelParams*, *sigma_scale=0.2*)

Fraction of positive tests $\phi_{\text{age},a}$.

$$\begin{aligned}\phi_{\text{age},a} &= e^{\phi^\dagger_{\text{age},a}}\\ \phi^\dagger_{\text{age},a} &= \mathcal{N}\left(0,\sigma_{\phi^\dagger_{\text{age},a}}\right)\\ \sigma_{\phi^\dagger_{\text{age},a}} &= HalfNormal\left(0.2\right)\end{aligned}$$

Parameters

- **name** (*str*) – Name of the variable $\phi_{\text{age},a}$. Will also appear in the trace with this name.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **sigma_scale** – Scale parameter for the HalfNormal distribution $\sigma_{\phi^\dagger_{\text{age},a}}$. 0.2

Returns $\phi_{\text{age},a}$ batch, age_group

`covid19_npis.model.number_of_tests._construct_reporting_delay`(*name*, *modelParams*, *m_ast*, *mu_loc=1.5*, *mu_scale=0.4*, *theta_sigma_scale=0.2*, *m_sigma_scale=3.0*)

$$m_{D_\text{test},c,b} = m^*_{D_\text{test},c,b} + \Delta m_{D_\text{test},c}$$

$$\begin{aligned}
\Delta m_{D_\text{test},c} &\sim \mathcal{N}(0, \sigma_{m_{D\text{ test}}}),\\
\sigma_{m_{D\text{ test}}} &\sim HalfNormal(3),\\
\theta_{D_\text{test},c} &\sim \mathcal{N}(\mu_{\theta_{D_\text{test}}}, \sigma_{\theta_{D_\text{test}}}),\\
\mu_{\theta_{D_\text{test}}} &\sim \mathcal{N}(1.5, 0.4),\\
\sigma_{\theta_{D_\text{test}}} &\sim HalfNormal(0.2).
\end{aligned}$$

Parameters

- **name** (*str*) – Name of the reporting delay variable $m_{D_\text{test},c,b}$.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **m_ast** (*tf.Tensor*) – $m^*_{D_\text{test},c,b}$ batch, country, spline
- **mu_loc** (*optional*) – Location parameter for the Normal distribution $\mu_{\theta_{D_\text{test}}}$. 1.5
- **mu_scale** (*optional*) – Scale parameter for the Normal distribution $\mu_{\theta_{D_\text{test}}}$. 0.4
- **theta_sigma_scale** (*optional*) – Scale parameter for the HalfNorml distribution $\sigma_{\theta_{D_\text{test}}}$. 0.2
- **m_sigma_scale** (*optional*) – Scale parameter for the HalfNorml distribution $\sigma_{m_{D\text{ test}}}$. 3.0

Returns $m_{D_\text{test},c,b}$ batch, country, spline

covid19_npis.model.number_of_tests.**_calc_reporting_delay_kernel**(*name*, *m*, *theta*, *length_kernel=14*)

Calculates the pdf for the gamma reporting kernel.

$$f_{c,t}(\tau) = Gamma(\tau; \alpha = \frac{m_{D_\text{test},c}(t)}{\theta_{D_\text{test}}, c} + 1, \beta = \frac{1}{\theta_{D_\text{test},c}}),$$

$$\text{with } f_{c,t} \text{ normalized such that } \sum_{\tau=0}^{T} f_{c,t}(\tau) = 1.$$

Parameters

- **name** – Name of the reporting delay kernel $f_{c,t}(\tau)$
- **m** – batch, time, country
- **theta** – batch, country
- **length_kernel** (*optional*) – Length of the kernel in days 14 days

Returns batch,country, kernel, time

covid19_npis.model.number_of_tests.**construct_testing_state**(*name_phi*, *name_eta*, *name_xi*, *name_m_ast*, *modelParams*, *num_knots*, *mu_cross_loc=0.0*, *mu_cross_scale=10.0*, *m_mu_loc=12.0*, *m_mu_scale=2.0*, *sigma_cross_scale=10.0*, *m_sigma_scale=1.0*)

$$(\phi^\dagger_{\text{tested},c,b},\ \eta^\dagger_{\text{traced},c,b},\ \xi^\dagger_{c,b},\ m^*_{D_\text{test},c,b}) \sim StudentT_{\nu=4}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$$

where

$$\boldsymbol{\mu} = \left(\mu_{\phi^\dagger_+}, \mu_{\eta^\dagger_{\text{traced}}}, \mu_{\xi^\dagger}, \mu_{m_{D_\text{test}}}\right)$$

$$\boldsymbol{\Sigma} \sim LKJ(\eta = 2, \boldsymbol{\sigma} = (\sigma_\phi, \sigma_\eta, \sigma_\xi, \sigma_m))$$

with the distributions parametarized in the following hierarchical manner:

$$\begin{aligned}
\mu_{\phi^\dagger_+},\ \mu_{\eta^\dagger_{\text{traced}}},\ \mu_{\xi^\dagger} &\sim \mathcal{N}(0, 10),\\
\mu_{m_{D_\text{test}}} &\sim \mathcal{N}(12, 2),\\
\sigma_\phi, \sigma_\eta, \sigma_\xi &\sim HalfCauchy(10),\\
\sigma_m &\sim HalfNormal(1)
\end{aligned}$$

at last we transform the variables $\phi_{+,c,b},\ \eta_{\text{traced},c,b},\ \xi_{c,b}$

$$\begin{aligned}
\phi_{+,c,b} &= \frac{e^{\phi^\dagger_{+,c,b}}}{e^{\phi^\dagger_{+,c,b}} + 1},\\
\eta_{\text{traced},c,b} &= \ln\left(1 + e^{\eta^\dagger_{\text{traced},c,b}}\right),\\
\xi_{c,b} &= \ln\left(1 + e^{\xi^\dagger_{c,b}}\right)\frac{n_{\text{inhabitants}}}{10000}
\end{aligned}$$

Parameters

- **name_phi** (*str*) – Name of the fraction of positive tests variable $\phi_{+,c,b}$.
- **name_eta** (*str*) – Name of the number of traced persons variable $\eta_{\text{traced},c,b}$.
- **name_xi** (*str*) – Name of the base tests rate variable $\xi_{c,b}$.
- **name_m_ast** (*str*) – Name of the testing delay variable $m^*_{D_{test},c,b}$.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **num_knots** – Number of knots for the Bspline dimension.
- **mu_cross_loc** (*optional*) – Location parameter for the three Normal distributions $\mu_{\phi^\dagger_+},\ \mu_{\eta^\dagger_{\text{traced}}},\ \mu_{\xi^\dagger}$. 0.0
- **mu_cross_scale** (*optional*) – Scale parameter for the three Normal distributions $\mu_{\phi^\dagger_+},\ \mu_{\eta^\dagger_{\text{traced}}},\ \mu_{\xi^\dagger}$. 10.0
- **m_mu_loc** (*optional*) – Location parameter for the Normal distribution $\mu_{m_{D_\text{test}}}$. 12.0
- **m_mu_scale** (*optional*) – Scale parameter for the Normal distribution $\mu_{m_{D_\text{test}}}$. 2.0
- **sigma_cross_scale** (*optional*) – Scale parameter for the three HalfCauchy distributions $\sigma_\phi, \sigma_\eta, \sigma_\xi$. 10.0
- **m_sigma_scale** (*optional*) – Scale parameter for the HalfNormal distribution σ_m. 1.0

Returns Testing state tuple $(\phi_{+,c,b},\ \eta_{\text{traced},c,b},\ \xi_{c,b},\ m_{D_\text{test},c,b}),\ \theta_{D_\text{test}}$. 4 x (batch, country, spline),

covid19_npis.model.number_of_tests.**construct_Bsplines_basis**(*modelParams*)

Function to construct the basis functions for all BSplines, should only be called once. Uses splipy python library.

Parameters

- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.

- **degree** (*optional*) – degree corresponds to exponent of the splines i.e. degree of three corresponds to a cubic spline. 3
- **knots** (*list, optional*) – Knots array used for constructing the BSplines. one knot every 7 days

Returns time, knots?

`covid19_npis.model.number_of_tests.`**`_calculate_Bsplines`**(*coef*, *basis*)

Calculates the Bsplines given the basis functions B and the coefficients x.

$$x(t) = \sum_b x_b B_b(t)$$

Parameters

- **coef** – Coefficients x. ...,country, spline
- **basis** – Basis functions tensor B. time, spline

Returns $x(t)$ ...,time, country

4.5 Deaths

`covid19_npis.model.deaths.`**`_construct_reporting_delay`**(*name*, *modelParams*, *theta_sigma_scale=0.3*, *theta_mu_loc=1.5*, *theta_mu_scale=0.3*, *m_sigma_scale=4.0*, *m_mu_loc=21.0*, *m_mu_scale=2.0*)

$$\begin{aligned}
m_{D_\text{death},c} &= \ln\left(1 + e^{m^*_{D_\text{death},c}}\right)\\
m^*_{D_\text{death},c} &\sim \mathcal{N}(\mu_{m_{D_\text{death}}}, \sigma_{m_{D_\text{death}}}),\\
\mu_{m_{D_\text{death}}} &\sim \mathcal{N}(21, 2),\\
\sigma_{m_{D_\text{test}}} &\sim HalfNormal(4),\\
\theta_{D_\text{death},c} &= \frac{1}{4}\ln\left(1 + e^{4\theta^*_{D_\text{death},c}}\right)\\
\theta^*_{D_\text{death},c} &\sim \mathcal{N}(\mu_{\theta_{D_\text{test}}}, \sigma_{\theta_{D_\text{test}}}),\\
\mu_{\theta_{D_\text{death}}} &\sim \mathcal{N}(1.5, 0.3),\\
\sigma_{\theta_{D_\text{death}}} &\sim HalfNormal(0.3).
\end{aligned}$$

Parameters

- **name** (*str*) – Name of the reporting delay variable $m_{D_\text{test},c,b}$.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **theta_sigma_scale** (*optional*) – Scale parameter for the Normal distribution $\sigma_{\theta_{D_\text{death}}}$. 0.3
- **theta_mu_loc** (*optional*) – Location parameter for the Normal distribution $\mu_{\theta_{D_\text{death}}}$. 1.5
- **theta_mu_scale** (*optional*) – Scale parameter for the HalfNormal distribution $\mu_{\theta_{D_\text{death}}}$. 0.3

- **m_sigma_scale** (*optional*) – Scale parameter for the HalfNormal distribution $\sigma_{m_{D_{\text{test}}}}$. 4.0
- **m_mu_loc** (*optional*) – Location parameter for the Normal distribution $\mu_{m_{D_{\text{death}}}}$. 21.0
- **m_mu_scale** (*optional*) – Scale parameter for the Normal distribution $\mu_{m_{D_{\text{death}}}}$. 2.0

Returns (m, theta) (batch, country) x 2

`covid19_npis.model.deaths.`**`_calc_Phi_IFR`**(*name, modelParams, alpha_loc=0.119, alpha_scale=0.003, beta_loc=-7.53, beta_scale=0.4*)

Calculates and construct the IFR and Phi_IFR:

$$\beta_{\text{IFR,c}} = \mathcal{N}(-7.53, 0.4)$$
$$\alpha_{\text{IFR}} = \mathcal{N}(0.119, 0.003)$$
$$\text{IFR}_c(a^*) = \frac{1}{100}\exp\left(\beta_{\text{IFR,c}} + \alpha_{\text{IFR}} \cdot a\right)$$
$$\phi_{\text{IFR},c,a} = \frac{1}{\sum_{a^*=a_{\text{beg}}(a)}^{a_{\text{end}}(a)} N_{\text{pop}}(a^*)} \sum_{a^*=a_{\text{beg}}(a)}^{a_{\text{end}}(a)} N_{\text{pop},c}(a^*)\, \text{IFR}_c(a^*),$$

Parameters

- **name** (*str*) – Name of the infection fatatlity ratio variable $\phi_{\text{IFR},c,a}$.
- **modelParams** (*covid19_npis.ModelParams*) – Instance of modelParams, mainly used for number of age groups and number of countries.
- **alpha_loc** (*optional*) – 0.119
- **alpha_scale** (*optional*) – 0.003
- **beta_loc** (*optional*) – -7.53
- **beta_scale** (*optional*) – 0.4

Returns Phi_IFR batch, country, age brackets

`covid19_npis.model.deaths.`**`calc_delayed_deaths`**(*name, new_cases, Phi_IFR, m, theta, length_kernel=40*)

Calculates delayed deahs from IFR and delay kernel.

$$\tilde{E}_{\text{delayDeath},c,a}(t) = \phi_{\text{IFR},c,a} \sum_{\tau=0}^{T} \tilde{E}_{c,a}(t-\tau) \cdot f_{c,t}(\tau)$$
$$f_{c,t}(\tau) = Gamma(\tau; \alpha = \frac{m_{D_{\text{death},c}}}{\theta_{D_{\text{death}}}} + 1, \beta = \frac{1}{\theta_{D_{\text{death}}}})$$

Parameters

- **name** (*str*) – Name of the delayed deaths variable $\tilde{E}_{\text{delayDeath},c,a}(t)$.
- **new_cases** (*tf.Tensor*) – New cases without reporting delay $\tilde{E}_{c,a}(t)$. batch, time, country, age_group
- **Phi_IFR** (*tf.Tensor*) – Infection fatality ratio of the age brackets $\phi_{\text{IFR},c,a}$. batch, country, age_group
- **m** (*tf.Tensor*) – Median fatality delay for the delay kernel $m_{D_{\text{death},c}}$. batch, country
- **theta** (*tf.Tensor*) – Scale fatality delay for the delay kernel $\theta_{D_{\text{death}}}$. batch
- **length_kernel** (*optional*) – Length of the kernel in days 14 days

Returns $\tilde{E}_{\text{delayDeath},c,a}(t)$ batch, time, country, age_group

4.6 Utility

`covid19_npis.model.utils.`**`gamma`**(*x*, *alpha*, *beta*)

Returns a gamma kernel evaluated at x. The implementation is the same as defined in the tfp.gamma distribution which is probably quiet numerically stable. :param x: :param alpha: :param beta:

`covid19_npis.model.utils.`**`positive_axes`**(*axes*, *ndim*)

Given a list of axes, returns them written as positive numbers

Parameters

- **axes** (*array-like, int*) – list of axes, positive or negative
- **ndim** (*int*) – number of dimensions of the array

Returns

Return type positive list of axes

`covid19_npis.model.utils.`**`match_axes`**(*tensor*, *target_axes*, *ndim=None*)

Extend and transpose dimensions, such that the dimension i of *tensor* is at the position target_axes[i]. Missing dimension are filled with size 1 dimensions. This is therefore a generalization of tf.expand_dims and tf.transpose and implemented using these. If ndim is None, the number of the dimensions of the result is the minimum fullfilling the requirements of target_axes

Parameters

- **tensor** (*tf.Tensor*) – The input tensor with len(tensor.dims) == len(target_axes)
- **target_axes** (*list of ints*) – Target positions of the dimensions. Can be negative.

Returns The transposed and expanded tensor.

Return type tensor

`covid19_npis.model.utils.`**`einsum_indexed`**(*tensor1*, *tensor2*, *inner1=()*, *inner2=()*, *outer1=()*, *outer2=()*, *vec1=()*, *vec2=()*, *targ_outer1=()*, *targ_outer2=()*)

Calling tf.einsum with indices instead of a string. For example einsum_indexed(t1, t2, inner1=1, inner2=0, outer1=0, outer2=1) corresponds to the *tf.einsum* string "ab…,bc…->ac…" (Matrix product) and a matrix vector product "…ab,…b,->…a" is parameterized by einsum_indexed(t1, t2, inner1=-1, inner2=-1, vec1=-2)

Parameters

- **tensor1** (*tensor*) – Input tensor 1
- **tensor2** (*tensor*) – Input tensor 2
- **inner1** (*int or list*) – The axes in tensor 1 over which a inner product is taken
- **inner2** (*int or list*) – The axes indices in tensor 2 over which a inner product is taken
- **outer1** (*int or list*) – The axes indices in tensor 1 over which a outer product is taken
- **outer2** (*int or list*) – The axes indices in tensor 2 over which a outer product is taken
- **vec1** (*int or list*) – The axes indices of the matrix in a matrix-vector product which are "staying" in the result. This is for the case where tensor1 corresponds to the matrix.
- **vec2** (*int or list*) – The axes indices of the matrix in a matrix-vector product which are "staying" in the result. This is for the case where tensor2 corresponds to the matrix.

- **targ_outer1** (*int or list*) – The axes indices in the result where the outer product axes of tensor 1 is mapped to. If omitted, the position is inferred such that the order stays the same, and, if equal, the indices of tensor 1 are to the left of the indices of tensor2 for outer products.
- **targ_outer2** (*int or list*) – The axes indices in the result where the outer product axes of tensor 2 is mapped to. If omitted, the position is inferred such that the order stays the same, and, if equal, the indices of tensor 1 are to the left of the indices of tensor2 for outer products.

Returns

Return type tensor

`covid19_npis.model.utils.`**`concatenate_axes`**(*tensor*, *axis1*, *axis2*)

Concatenates two consecutive axess

Parameters

- **tensor** (*tensor*) – input
- **axis1** (*int*) – first axis
- **axis2** (*int*) – second axis

Returns

Return type Concatenated tensor

`covid19_npis.model.utils.`**`slice_of_axis`**(*tensor*, *axis*, *begin*, *end*)

Returns the tensor where the axis *axis* is sliced from *begin* to *end*

Parameters

- **tensor** (*tensor*) –
- **axis** (*int*) –
- **begin** (*int*) –
- **end** (*int*) –

Returns

Return type sliced tensor

`covid19_npis.model.utils.`**`convolution_with_fixed_kernel`**(*data*, *kernel*, *data_time_axis*, *filter_axes_data=()*)

Convolve data with a time independent kernel. The returned shape is equal to the shape of data. In order avoid constructing a time_length x time_length kernel, the data is decomposed in overlapping frames, with a stride of *padding*, allowing to construct a only padding x time_length sized kernel.

Parameters

- **data** (*tensor*) – The input tensor
- **kernel** (*tensor*) – Has as shape filter_axes x time. filter_axes can be several axes, where in each dimension a difference kernel is located
- **data_time_axis** (*int*) – the axis of data which corresponds to the time axis
- **filter_axes_data** (*tuple*) – the axes of *data*, to which the *filter_axes* of *kernel* should be mapped to. Each of this dimension is therefore subject to a different filter

Returns

Return type A convolved tensor with the same shape as data.

`covid19_npis.model.utils.`**`convolution_with_varying_kernel`**(*data*, *kernel*, *data_time_axis*, *filter_axes_data=()*)

Convolve data with a time dependent kernel. The returned shape is equal to the shape of data. In this implementation, the kernel will be augmented by a time_data axis, and then the inner product with the date will be taken. This is not an optimal implementation, as the most of the entries of the kernel inner product matrix will be zero.

Parameters

- **`data`** (*`tensor`*) – The input tensor
- **`kernel`** (*`tensor`*) – Has as shape filter_axes x time_kernel x time_data. filter_axes can be several axes, where in each dimension a difference kernel is located
- **`data_time_axis`** (*`int`*) – the axis of data which corresponds to the time axis
- **`filter_axes_data`** (*`tuple`*) – the axes of *data*, to which the *filter_axes* of *kernel* should be mapped to. Each of this dimension is therefore subject to a different filter

Returns

Return type A convolved tensor with the same shape as data.

`covid19_npis.model.utils.`**`convolution_with_map`**(*data*, *kernel*, *modelParams*)

Parameters **`data`** – batch, time, country, agegroup

`covid19_npis.model.utils.`**`get_filter_axis_data_from_dims`**(*ndim*)

Returns filter axis data from len(new_I_t.shape)

CHAPTER FIVE

PLOT

There are multiple stages involved before one can start to plot the obtained data.

- For model description, see *Model*.
- Trace data can be converted with `covid19_npis.data.convert_trace_to_pandas_list()`.
- Plotting WIP

5.1 Data converter

`covid19_npis.da`

Converts the pymc4 arviz trace to multiple pandas dataframes. Also sets the right labels for the dimensions i.e splits data by country and age group.

Do not look too much into this function

if you want to keep your sanity!

Parameters

- **trace** (*arivz InferenceDa* –
- **sample_stat** (*pymc4 sample state*) –

Returns
Multiindex dataframe containing all samples by chain and other dimensions de-

fined in config.py

Return type
list of pd.DataFrame

`covid19_npis.da`
Converts the pymc4 arviz trace for a

single key to a pandas dataframes. Also sets the right labels for the dimensions i.e splits data by country and age group.

Do not look too much into this function if you want to keep your sanity!

Parameters

- **trace** (*arivz InferenceDa*) –
- **sample_stat** (*pymc4 sample state*) –
- **key** (*str*) – Name of variable in model-

Params

- **data_type** (*str*) – Type of trace, gets detected automatically normally.

Possible values are: "posterior", "prior_predictive", "posterior_predictive". Overwrites automatic behaviour! default: None

Returns
Multiindex dataframe containing all samples by chain and other dimensions defined in modelParams.py

Return type
pd.DataFrame

5.2 Distri

`covid19_npis.pl`

High level plotting function for dis- tri- bu- tions, plots prior and pos- te- rior if they are given in the trace.

Parameters

- **trace** (*arivz. InferenceDa* – Raw data from pymc4 sam- pling, can con- tain both pos- te- rior data and prior data. Or only one of both!
- **sample_stat** (*pymc4 sample state*) – Used mainly for shape la- bels
- **key** (*str*) – Name of the vari-

able to plot

- **dir_save** (*str*, *optional*) – where to save the the figures (expecting a

folder). Does not save if None None

- **force_matr** (*bool*, *optional*) – Forces matrix plotting behaviour for last two di-

mensions False

Returns
one figure for each country

Return type
array of mpl fig-

ures

`covid19_npis.pl`

High level function to create a distribution plot for matrix like variables e.g. ‘C’. Uses last two dimensions for matrix like plot.

Parameters

- **trace** (*arivz. InferenceDa* – Raw data from pymc4 sampling, can contain both posterior data and prior data. Or only one of both!
- **sample_stat** (*pymc4 sample state*) – Used mainly for shape labels

- **key** (*str*) – Name of the variable to plot
- **dir_save** (*str*, *optional*) – where to save the the figures (expecting a

folder). Does not save if None None

Returns

Return type
axes

`covid19_npis.pl`

Low level function to plots posterior and prior from arrays.

Parameters

- **array_prior** (*array_post*) – Sampling data as array, should be filtered beforehand. If none it does not get plotted!
- **dist_name** (*str*) – name of distribution for plotting
- **dist_math** (*str*) – math of distribution for plotting
- **suffix** (*str, optional*) – Suffix for

the plot ti- tle e.g. “age_group_1” “”

- **ax** (*mpl axes element, optional*) – Plot into an ex- ist- ing axes el- e- ment None

covid19_npis.pl

Low level plot- ting func- tion, plots the prior as line for sam- pling data by using kernel density estimation. For more references see scipy documentation.

It is highly rec- om- mended to pass an

axis otherwise the xlim may be a bit wonky.

Parameters

- **x** – Input values, from sampling
- **ax** (*mpl axes element, optional*) – Plot into an existing axes element None
- **kwargs** (*dict, optional*) – Directly passed to plotting mpl.

`covid19_npis.pl`

Low level plotting function to plot an sampling data as histogram.

Parameters

- **x** – Input values, from sampling
- **bins** (*int*, *optional*) – Defines the number of equal-width bins in the range. 50
- **ax** (*mpl axes element*,

optional) – Plot into an existing axes element None

- **kwargs** (*dict*, *optional*) – Directly passed to plotting mpl.

`covid19_npis.pl`
helper to get coordinates of a text object in the coordinates of the axes element [0,1]. used for the rectangle backdrop.

Returns: x_min, x_max, y_min, y_max

`covid19_npis.pl`

add a rect- an- gle to the axes (be- hind the text)

provide a list of text el- e- ments and pos- si- ble op- tions passed to mpl.patches.Re

e.g. facecolor="grey", alpha=0.2, zorder=99,

5.3 Time se- ries

`covid19_npis.pl`

High level plot- ting fuc- n- tion to cre- ate time series for a a give variable, i.e. plot for every additional dimension. Can only be done for variables with a time or date in shape_labels!

Does NOT plot observed cases, these have to be added manually for now.

Parameters

- **trace_prior** (*trace_post*) – Raw data from pymc4 sampling
- **sample_stat** (*pymc4 sample stae*) –
- **key** (*str*) – Name of the time-series

variable to plot. Same name as in the model definitions.

- **sampling_ty** (*str*, *optional*) – Name of the type (group) in the arviz inference data. posterior

- **dir_save** (*str*, *optional*) – where to save the the figures (expecting a folder). Does not save if None None

- **observed** (*pd. DataFrame*, *optional*) –

mod- el- Params dataframe for the cor- re- spond- ing

observed values for the variable e.g. modelParams.pos_tests_dataframe

`covid19_npis.pl`

low- level func- tion to plot any- thing that has a date on the x- axis.

Parameters

- **x** (*array of datetime. datetime*) – times for the x axis
- **y** (*array, 1d or 2d*) –

data to plot. if 2d, we plot the CI as fill_between (if CI enabled in rc params) if 2d, then first dim is realization and second dim is time matching *x* if 1d then first tim is time matching *x*

- **ax** (*mpl axes element, optional*) – plot into an existing axes element. default: None
- **what** (*str, optional*) – what type of data is provided in x. sets the style used for plotting: * *data* for data points * *fcast* for model forecast (prediction) * *model* for model reproduction of data (past)
- **kwargs** (*dict,*

optional) – directly passed to plotting mpl.

Returns

Return type
ax

CHAPTER SIX

DATA & MODELPARAMS

We apply a utility/abstraction layer to our data to simplify plotting and other operations later.

Before we can construct our model Parameters, we have to import and manipulate/restructure our data a bit as follows:

6.1 Data

class covid19_r

Country data class! Contains death, new_cases/posi tests, daily tests, interventions and config data for a specific country. Retrieves this data from a gives folder. There are the following specifications for the data:

- **new_cases.csv**

- Time/Date column has to be named "date" or "time"
- Age group columns have to be named consistent between different data and countries

- **interventions.csv**
  - Time/Date column has to be named "date" or "time"
  - Different intervention as ad-

di- tional columns with in- ter- ven- tion name as col- umn name

- **tests.csv**
  - Time/Date col- umn has to be named “date” or “time”
  - Daily per- formed tests col- umn with name “tests”
- **deaths.csv**
  - Time/Date col- umn has to be named “date” or “time”
  - Daily

deaths column has to be named "deaths"

- Optional: Daily deaths per age group same column names as in new_cases

- **population.csv**
  - Age column named "age"
  - Column Number of people per age named "PopTotal"
- **config.json, dict:**
  - name : "country_name"

– **age_groups** [dict]

  * “column_name” : [age_lower, age_upper]

  Also calculates change points and interventions automatically on init.

Parameters
  path_to_fol (*string*) – Filepath to the folder, which holds all the data for the country! Should be something like “../data/German That is new_cases.csv, in-

ter- ven- tions.csv,

population.csv

create_change_

Create change points for a single intervention and also adds interventions if they do not exist yet.

Parameters

df (*pandas.DataFrame*) – Single intervention column with datetime index.

Returns

Change points dict

{name:[cps

classmethod ad

Constructs and adds in- ter- ven- tion to the class at- tributes if that in- ter- ven- tion does not ex- ist yet! This is done by name check.

Parameters

- **name** (*string*) – Name of the in- ter- ven- tion
- **time_series** (*pandas. DataFrame*) – In- ter- ven-

tion in- dexs as time se- ries with date- time in- dex!

classmethod se

Manual set prior for ef- fec- tiv- ity al- pha for a in- ter- ven- tion via the name. That is it set prior_alpha_loc and prior_alpha_sca of a In- tervention instance.

Parameters

- **name** (*string*) – Name of in-

ter-
ven-
tion

- **prior_loc** (*number*) –
- **prior_scale** (*number*) –

classmethod ge

Gets in- ter- ven- tion from in- ter- ven- tions ar- ray via name

Returns
Intervention

class covid19_r

Parameters

- **name** (*string*) – Name of the in- ter- ven- tion
- **num_stages** (*int*,) – Num- ber

of different stages the intervention can have.

- **prior_alpha** (*number, optional*) –
- **prior_alpha** (*number, optional*) –

class covid19_n

Parameters

- **prior_date_** (*number*) – Mean of prior distribution for the location (date) of the change point.
- **gamma_max**

– Gamma max value for change point

- **length** (*number, optional*) – Length of change point
- **prior_date_** (*number, optional*) – Scale of prior distribution for the location (date) of the change point.

6.2 Mode

class covid19_r

This is a class for

all model parameters. It is mainly used to have a convenient to access data in model wide parameters e.g. start date for simulation.

This class also contains the data used for fitting. *dataframe* is the original dataframe. *data_tensor* is a tensor in the correct shape (time x countries x age) with values replace by nans when no data is available.

Parameters
countries (list, *covid19_np data. Country*) – Data objects for multiple countries

classmethod fr
Create modelParams class from folder containing differet regions or countrys

property count
Data objectes for each country.

Returns
List of all country ob-

ject

_update_data_s

Update Data summary

property data_

Data summary for model-Params object.

property gamma

Creates a ragged tensor with dimension intervention, country, change_points The change points dimension can have different sizes.

property date_

Creates a tensor

with dimension intervention, country, change_points Padded with 0.0 for none existing change points

property pos_t

New cases as multi-Column dataframe level 0 = country/region and level 1 = age group.

property pos_t

Numpy Array of daily new cases / positive tests for coun- tries/regions and age groups.

Returns
time, coun- try, age- group

Return type
tf.Tensor

property pos_t
Tensor of daily new cases / pos- i- tive tests for coun- tries/regions and age groups.

Returns
time, coun- try, age- group

Return type
tf.Tensor

property total
Dataframe of to- tal tests in all coun- tries. Date-

time in- dex and coun- try columns as Mul- ti- in- dex.

property total

returns: time, coun- try :rtype: tf.Tensor

property death

Dataframe of deaths in all coun- tries. Date- time in- dex and coun- try columns as Mul- ti- in- dex.

property death

returns: time, coun- try :rtype: tf.Tensor

property N_dat

Dataframe of pop- ula- tion in all coun- tries. Date- time in- dex and coun- try columns as Mul- ti- in- dex.

property N_dat

Creates the pop- u- la- tion ten- sor with au- to- mat- i- cally cal- cu- lated age strata/brackets. coun- try, age_groups

property N_dat

Creates the pop- u- la- tion ten- sor for every age. coun- try, age

property indic

Returns the in- dex of ev- ery coun- try when the first case is re- ported. It could be that for some coun- tries, the in- dex is later than self.offset_sim_data.

property lengt

returns: Length of the in- serted/loaded data in days :rtype: num- ber

property lengt

returns: Length

of the sim- u- la- tion in days. :rtype: num- ber

property splin

Calculates B- spline ba- sis.

Returns

Return type

modelParams.le mod- el- Params.num_sp

_make_global()

Run once if you want to make the mod- el- Params global. Used in plot- ting

CHAPTER

SEVEN

CONTRIBUTING

7.1 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 have a config in the repository that uses black as a pre-commit hook.

This snippet should get you up and running:

```
conda install -c conda-forge black
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.

7.2 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 :)

7.3 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 required python packages for the documentation and compile the documentation.

```
cd docs
pip install -r piprequirements.txt
make html
```

The documentation can now be be accessed locally in `docs/_build/html/index.html`. As an example for the docstring formatting you can look at the documentation of `covid19_npis.model.disease_spread()`. We try to use the numpydoc style.

CHAPTER EIGHT

DEBUGGING

This is a small list of debug code snippets.

8.1 Debugging nans with tensorflow

It is a little problematic, because some nans occur during the runtime without being an error. Often these are cases where an operation has different implementations based on the value of the input, because it would otherwise lead to a loss of precision.

Therefore we wrote some patches, which put try-except blocks around these code parts and if a error occurs, disable check_numeric for this part.

For patching tensorflow_probability (replace the variables by the correct path):

```
cd scripts/debugging_patches
patch -d {$CONDA_PATH}/envs/{$ENVIRONMENT_NAME}/ -p 0 < filter_nan_errors1.patch
patch -d {$CONDA_PATH}/envs/{$ENVIRONMENT_NAME}/ -p 0 < filter_nan_errors2.patch
patch -d {$CONDA_PATH}/envs/{$ENVIRONMENT_NAME}/ -p 0 < filter_nan_errors3.patch
```

And then uncomment these line of codes in the run_script. Check_numerics has to enabled only before the optimization, not before the initial sampling, because the nan occuring during the sampling of the gamma distribution hasn't been patched.

```
tf.config.run_functions_eagerly(True)
tf.debugging.enable_check_numerics(stack_height_limit=30, path_length_limit=50)
```

For debugging the VI, it is reasonable to increase the step size, to run more quickly into errors

8.2 Basic usage of logger

```
# Change to debug mode i.e all log.debug is printed
logging.basicConfig(level=logging.DEBUG)

# Use log.debug instead of print
log.debug(f"My var {var}")
```

8.3 Force cpu or other device

```
my_devices = tf.config.experimental.list_physical_devices(device_type="CPU")
tf.config.experimental.set_visible_devices(devices=my_devices, device_type="CPU")
tf.config.set_visible_devices([], "GPU")
```

CHAPTER

NINE

HOW TO BUILD A DATASET FOR OUR MODEL

To use our model you may want to create your own dataset. In the following we try to guide you through the process of creating your own dataset. Feel free to take a look into our script. we use to create our dataset.

We use a hierarchical for our data as for our model. To add new country or region to our model we first create a folder containing the data.

```
mkdir test_country
```

Next we create a config.json file inside this folder. The json has to contain a unique name for the country/region and the age group brackets. You can add any number of age groups, the name of the groups should be the same across all countries though! We use four different age groups for most of our analysis as follows.

```
{
    "name": "test_country",
    "age_groups": {
            "age_group_0" : [0,34],
            "age_group_1" : [35,59],
            "age_group_2" : [60,79],
            "age_group_3" : [80,100]
    }
}
```

- **config.json, dict:**
    - name : “country_name”
    - **age_groups** [dict]
        - “column_name” : [age_lower, age_upper]

9.1 Population data

Each dataset for a country/region needs to contain population data for every age from 0 to 100. The data should be saved as population.csv! Most of the population data can be found on the UN website.

age	PopTotal
0	831175
1	312190
…	…

- Age column named “age”
- Column Number of people per age named “PopTotal”

9.2 New cases/ Positive tests data

We supply the number of positive tested persons per day and age group as a csv file for our country/region. The file has to be named “new_cases.csv” and has to contain the same column names as defined in the config.json! That is the age groups. Date Format should be “%d.%m.%y”.

date	age_group_0	age_group_1	age_group_2	age_group_3
01.01.20	103	110	13	130
02.01.20	103	103	103	103
…	…	…	…	…

- Time/Date column has to be named “date” or “time”
- Age group columns have to be named consistent between different data and countries!

9.3 Total tests data

The number of total tests performed per day in the country/region is also supplied as a csv file called “tests.csv”. The format should be as follows:

date	tests
01.01.20	10323
02.01.20	13032
…	…

- Time/Date column has to be named “date” or “time”
- Daily performed tests column with name “tests”

9.4 Number of deaths data

The number of deaths per day in the country/region also supplied as csv file nameed “deaths.csv”.

date	deaths
01.01.20	10
02.01.20	35
…	…

- Time/Date column has to be named “date” or “time”
- Daily deaths column has to be named “deaths”
- Optional(not working yet): Daily deaths per age group same column names as in new_cases

9.5 Interventions data

The intervention is also added as csv file. The file has to be named "interventions.csv" and can contain any number of interventions. We use the the oxford response tracker for this purpose, but you can also construct your own time series.

You can call/name the interventions whatever you like. The index should be an integer though.

date	school_closing	cancel_events	curfew	...
01.01.20	1	0	0	...
02.01.20	1	0	0	...
03.01.20	1	2	3	...
04.01.20	2	2	3	...
05.01.20	2	1	0	...
...	...	...	...	...

- Time/Date column has to be named "date" or "time"
- Different intervention as additional columns with intervention name as column name

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX