Interactive CoViD-19 dashboard

David Heider

22 3 2020

Models and dashboards for forecasting CoViD-19 cases

The following is a report about an interactive dashboard and a model to model, forecast and visualize the spatio-temporal dynamics of the CoViD-19 pandemics. The figure below shows for the (default) country Germany what the model and corresponding code does.

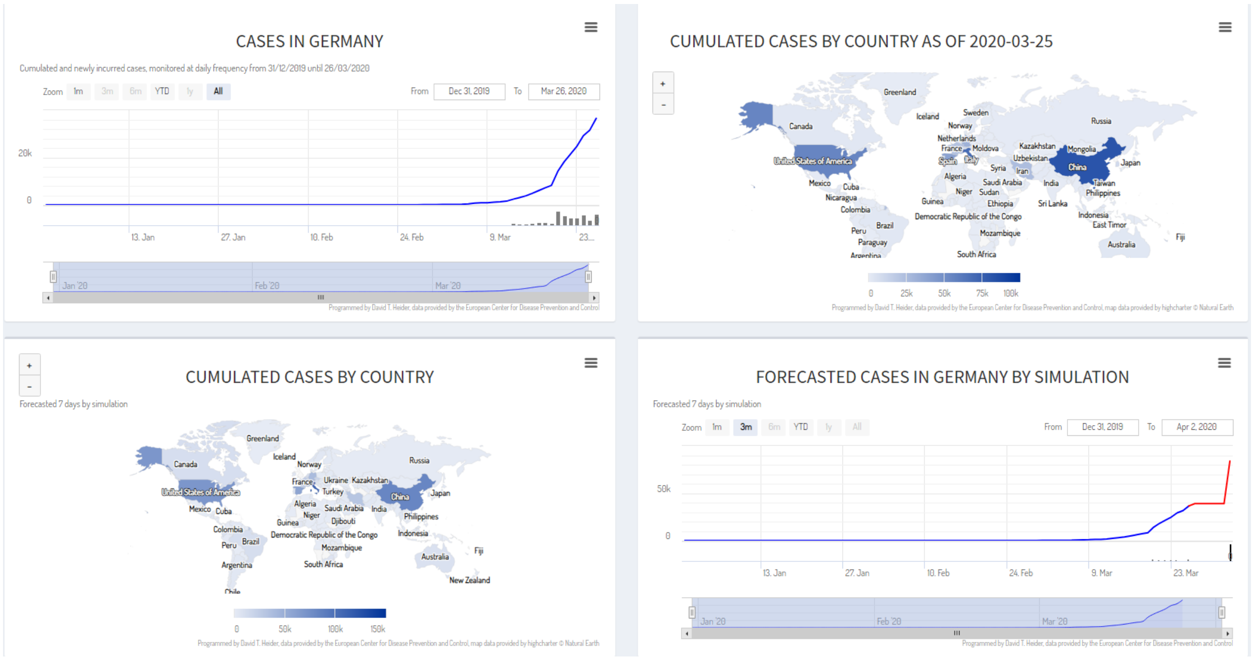

CASES IN GERMANY
Cumulated and newly incurred cases, monitored at daily frequency from 31/12/2019 until 26/03/2020
CUMULATED CASES BY COUNTRY AS OF 2020-03-25
CUMULATED CASES BY COUNTRY
Forecasted 7 days by simulation
FORECASTED CASES IN GERMANY BY SIMULATION
Forecasted 7 days by simulation
Programmed by David T. Heider, data provided by the European Center for Disease Prevention and Control

Figure 1: Empirical and forecasted dynamics of CoViD-19 on an interactive map (off-diagonal) and the temporal dynamics of the CoViD-19 pandemic in Germany (diagonal)

Content and theory

This dashboard presents the results of a simple population dynamics transferred to epidemiology, more precisely, to the study of the spread of the CoViD-19 virus over the globe. Let $n \in \mathbb{N}$ denote the number of countries available in (I) the CoViD-dataset provided by the ECDC, (II) the population dataset available as a JSON on github and (III) the countries listed in the geodatasource dataset that provides pairs of geographically neighboring countries. We consider $\mathbf{N} = \sum_{i=1}^{n} N_i \hat{e}_i \in \left(\mathbb{R}_0^+\right)^n$ as a representation of the number of people $N_i = N_i(t)$ in country i that have been infected with the virus up to time $t \geq 0$, measured in days. Note that they might have recovered or died at time $T \leq t$, yet are counted.

The model we investigate is

$$\frac{dN_i}{dt} = (P_i - N_i(t)) \cdot \langle \mathbf{k}_i, \mathbf{N} \rangle,$$

where P_i indicates the respective country's population at time $t = 0$. This is a generalization of the logistic growth model to include interactions in the vectorial constants. The $\mathbf{k}_i \in \mathbb{R}^n$, $1 \leq i \leq n$, can be bound by columns so as to give rise to a matrix of parameters, $K \in \mathbb{R}^{n^2}$.

Practically, the parameter matrix is determined by setting $K_{ij} = \mathbf{k}_i \cdot \hat{e}_j := 0$ whenever country i and country j do not share a geographic border according to the dataset (III). Otherwise, K_{ij} is determined via linear regression for each i, $1 \leq i \leq n$, individually to the epidemiological data provided by the ECDC. Numerically, $d_t N_i(t)$ is identified with the number of new CoViD-cases in the country at a given time t. The latter is measured in units of days from 2019-12-31 onwards. $N_i(t)$ is obtained from the data by taking the cumulative sum of the new CoViD-cases up to time t, inclusively. For details see the source code below. Since the regression model is linear in the matrix K, the estimate obtained that way is unbiased. The consistency could not be assessed thoroughly because the number of available datapoints was too small.

Ultimately, a time-forward simulation is conducted via the finite-differences method for systems of ordinary differential equations. The method is expounded in the usual suspects for textbooks on numerical mathematics. The empirical values of $N_i(T)$ serve as initial condition and dt is approximated by a time-interval of 1 day, as in the original data.

The model has been fitted for both reported quantities, the reported CoViD-19 cases and deaths. The results are shown graphically together with the fitting parameters in tab 2 and 3 of the dashboard. The figure in the bottom left corner of tab 1 depicts a binary view on the adjacency matrix which indicates in blue if two countries are neighboring. The white pixels indicate that the pair of countries listed as column and axis label share no geographic border in the sense of dataset (III).

The top row in tabs 1 and 2 depict the original data as a time series for a given country and the spatial distribution at a given point in time for which data has been gathered by the ECDC and made public. The middle row depicts the spatial forecast Δt days into the future, starting from the latest observation at $t = T$ as well as a time series plot for the empirical data (blue) and the simulated data (red) from $t = T + 1$ to $t = T + \Delta t$, i.e,, the forecast horizon. The bottom row depicts the entries of the parameter matrix $K = (K_{ij})_{1 \leq i,j \leq n}$ in the form of a scaled level plot. The matrix has been transformed by applying the mapping

$$\Pi : \mathbb{R}^2 \to \mathbb{R}^2,\ K_{ij} \mapsto \log \left| \frac{K_{i,j}}{\min_{(i,j)}\{K_{ij} | K_{ij} \neq 0\}} \right|$$

whenever $K_{ij} \neq 0$. If $K_{ij} = 0$, a null-continuation of Π has been used, i.e, $0 \mapsto 0$. Finally, the data grid in the bottom right corner depicts the numerical values of the parameter matrix K used in the simulation visualized in the middle row.

Based on the results of this project, the model's predictions should be treated as possibly reliable for a forecast horizon of at maximum 14 days from the date of the latest observation dataset (I). Obvious limitations on the number of available datapoints prohibit further inference on the dynamics of the virus pandemy in the context of the model. For a considerably short forecast hoirzon of less than 7 days, the forecast seems plausible.

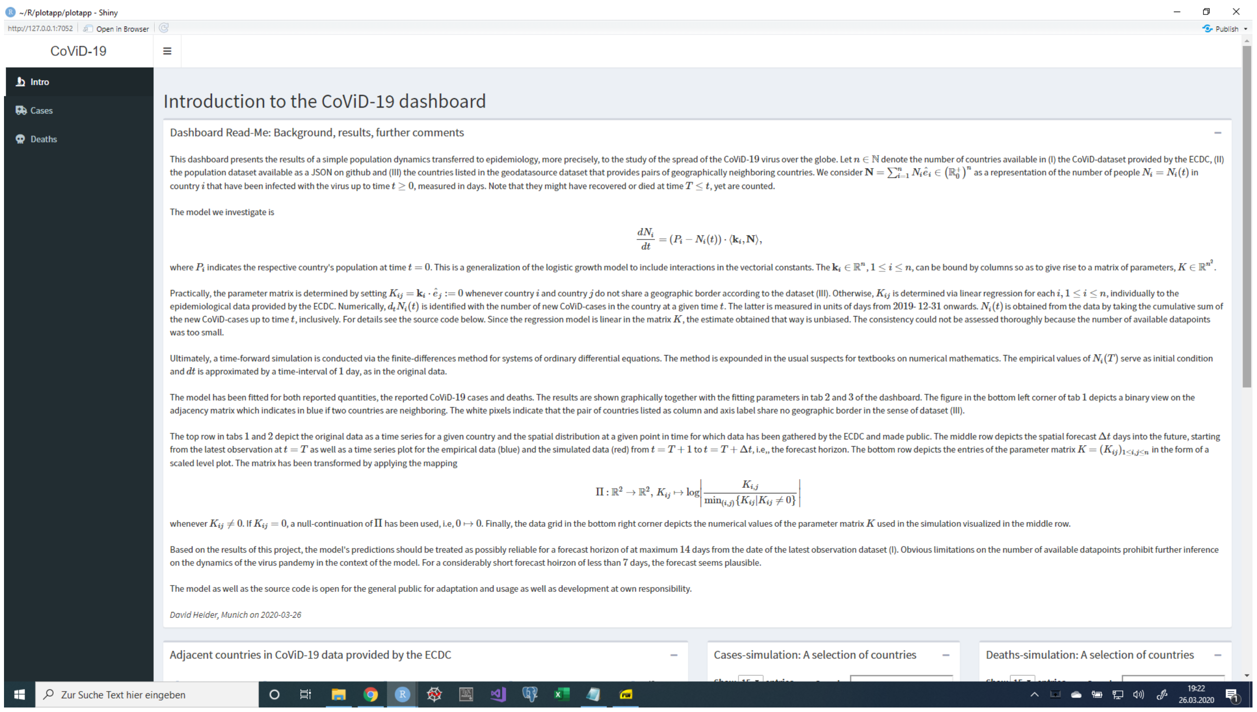
The model as well as the source code is open for the general public for adaptation and usage as well as development at own responsibility.

Snapshots of the dashboard and of the simulation

In the following, the dashboard is presented in the form of screenshots. The simulation tab for cumulated deaths has the same layout as the simulation tab for cumulated cases. Thus, only the former will be presented.

Simulation code

This section displays the code that is constitutes the R part of the application. The simulation, the import of relevant packages is done here. For pedagogic reasons, an individual section was devoted to the Shiny dashboard and its inner life. This way, readers may better choose which pages to read or print out.



Introduction to the CoViD-19 dashboard

Dashboard Read-Me: Background, results, further comments

This dashboard presents the results of a simple population dynamics transferred to epidemiology, more precisely, to the study of the spread of the CoViD-19 virus over the globe. Let $n \in \mathbb{N}$ denote the number of countries available in (I) the CoViD-dataset provided by the ECDC, (II) the population dataset available as a JSON on github and (III) the countries listed in the geodatasource dataset that provides pairs of geographically neighboring countries. We consider $\mathbf{N} = \sum_{i=1}^{n} N_i \hat{e}_i \in (\mathbb{R}_0^+)^n$ as a representation of the number of people $N_i = N_i(t)$ in country i that have been infected with the virus up to time $t \geq 0$, measured in days. Note that they might have recovered or died at time $T \leq t$, yet are counted.

The model we investigate is

$$\frac{dN_i}{dt} = (P_i - N_i(t)) \cdot \langle \mathbf{k}_i, \mathbf{N} \rangle,$$

where P_i indicates the respective country's population at time $t = 0$. This is a generalization of the logistic growth model to include interactions in the vectorial constants. The $\mathbf{k}_i \in \mathbb{R}^n, 1 \leq i \leq n$, can be bound by columns so as to give rise to a matrix of parameters, $K \in \mathbb{R}^{n^2}$.

Practically, the parameter matrix is determined by setting $K_{ij} = \mathbf{k}_i \cdot \hat{e}_j := 0$ whenever country i and country j do not share a geographic border according to the dataset (III). Otherwise, K_{ij} is determined via linear regression for each $i, 1 \leq i \leq n$, individually to the epidemiological data provided by the ECDC. Numerically, $d_t N_i(t)$ is identified with the number of new CoViD-cases in the country at a given time t. The latter is measured in units of days from 2019- 12-31 onwards. $N_i(t)$ is obtained from the data by taking the cumulative sum of the new CoViD-cases up to time t, inclusively. For details see the source code below. Since the regression model is linear in the matrix K, the estimate obtained that way is unbiased. The consistency could not be assessed thoroughly because the number of available datapoints was too small.

Ultimately, a time-forward simulation is conducted via the finite-differences method for systems of ordinary differential equations. The method is expounded in the usual suspects for textbooks on numerical mathematics. The empirical values of $N_i(T)$ serve as initial condition and dt is approximated by a time-interval of 1 day, as in the original data.

The model has been fitted for both reported quantities, the reported CoViD-19 cases and deaths. The results are shown graphically together with the fitting parameters in tab 2 and 3 of the dashboard. The figure in the bottom left corner of tab 1 depicts a binary view on the adjacency matrix which indicates in blue if two countries are neighboring. The white pixels indicate that the pair of countries listed as column and axis label share no geographic border in the sense of dataset (III).

The top row in tabs 1 and 2 depict the original data as a time series for a given country and the spatial distribution at a given point in time for which data has been gathered by the ECDC and made public. The middle row depicts the spatial forecast Δt days into the future, starting from the latest observation at $t = T$ as well as a time series plot for the empirical data (blue) and the simulated data (red) from $t = T + 1$ to $t = T + \Delta t$, i.e,, the forecast horizon. The bottom row depicts the entries of the parameter matrix $K = (K_{ij})_{1 \leq i,j \leq n}$ in the form of a scaled level plot. The matrix has been transformed by applying the mapping

$$\Pi : \mathbb{R}^2 \to \mathbb{R}^2,\ K_{ij} \mapsto \log \left| \frac{K_{i,j}}{\min_{(i,j)} \{K_{ij} | K_{ij} \neq 0\}} \right|$$

whenever $K_{ij} \neq 0$. If $K_{ij} = 0$, a null-continuation of Π has been used, i.e, $0 \mapsto 0$. Finally, the data grid in the bottom right corner depicts the numerical values of the parameter matrix K used in the simulation visualized in the middle row.

Based on the results of this project, the model's predictions should be treated as possibly reliable for a forecast horizon of at maximum 14 days from the date of the latest observation dataset (I). Obvious limitations on the number of available datapoints prohibit further inference on the dynamics of the virus pandemy in the context of the model. For a considerably short forecast hoirzon of less than 7 days, the forecast seems plausible.

The model as well as the source code is open for the general public for adaptation and usage as well as development at own responsibility.

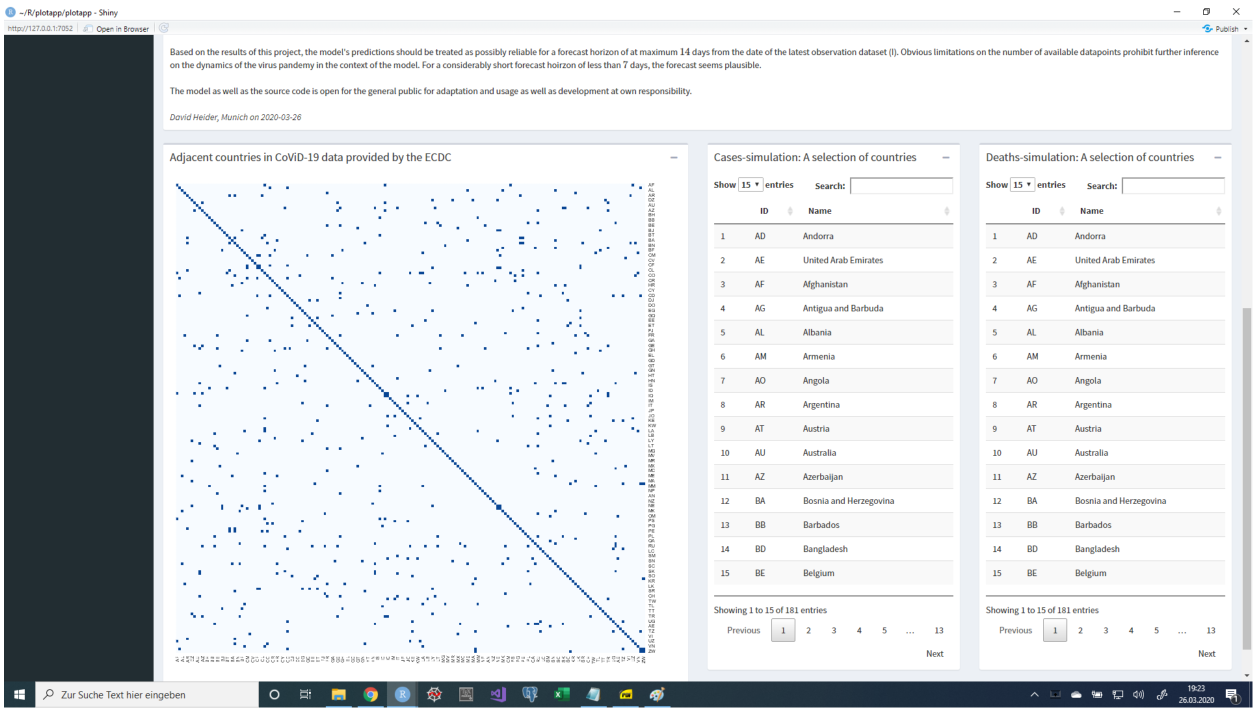
David Heider, Munich on 2020-03-26

Adjacent countries in CoViD-19 data provided by the ECDC

Cases-simulation: A selection of countries

Deaths-simulation: A selection of countries

Figure 2: Introduction tab - Top row: Theoretical background, cf. the first section in this PDF



Based on the results of this project, the model's predictions should be treated as possibly reliable for a forecast horizon of at maximum 14 days from the date of the latest observation dataset (I). Obvious limitations on the number of available datapoints prohibit further inference on the dynamics of the virus pandemy in the context of the model. For a considerably short forecast hoirzon of less than 7 days, the forecast seems plausible.

The model as well as the source code is open for the general public for adaptation and usage as well as development at own responsibility.

David Heider, Munich on 2020-03-26

Adjacent countries in CoViD-19 data provided by the ECDC

Cases-simulation: A selection of countries

Show 15 entries Search:

	ID	Name
1	AD	Andorra
2	AE	United Arab Emirates
3	AF	Afghanistan
4	AG	Antigua and Barbuda
5	AL	Albania
6	AM	Armenia
7	AO	Angola
8	AR	Argentina
9	AT	Austria
10	AU	Australia
11	AZ	Azerbaijan
12	BA	Bosnia and Herzegovina
13	BB	Barbados
14	BD	Bangladesh
15	BE	Belgium

Showing 1 to 15 of 181 entries

Previous 1 2 3 4 5 … 13 Next

Deaths-simulation: A selection of countries

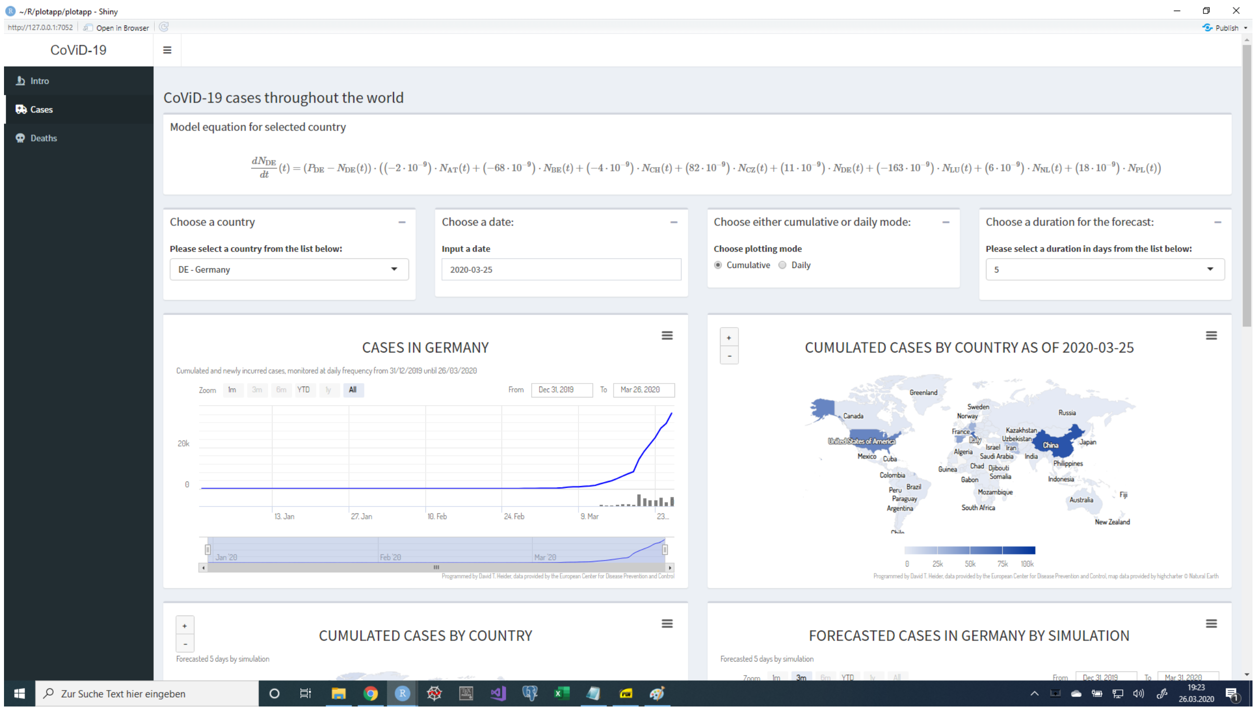
Show 15 entries Search:

	ID	Name
1	AD	Andorra
2	AE	United Arab Emirates
3	AF	Afghanistan
4	AG	Antigua and Barbuda
5	AL	Albania
6	AM	Armenia
7	AO	Angola
8	AR	Argentina
9	AT	Austria
10	AU	Australia
11	AZ	Azerbaijan
12	BA	Bosnia and Herzegovina
13	BB	Barbados
14	BD	Bangladesh
15	BE	Belgium

Showing 1 to 15 of 181 entries

Previous 1 2 3 4 5 … 13 Next

Figure 3: Introduction tab - Bottom row: Information about the geographic interdepdencies between countries struck with CoViD


CoViD-19

CoViD-19 cases throughout the world

Model equation for selected country

$$\frac{dN_{\mathrm{DE}}}{dt}(t) = (P_{\mathrm{DE}} - N_{\mathrm{DE}}(t)) \cdot ((-2 \cdot 10^{-9}) \cdot N_{\mathrm{AT}}(t) + (-68 \cdot 10^{-9}) \cdot N_{\mathrm{BE}}(t) + (-4 \cdot 10^{-9}) \cdot N_{\mathrm{CH}}(t) + (82 \cdot 10^{-9}) \cdot N_{\mathrm{CZ}}(t) + (11 \cdot 10^{-9}) \cdot N_{\mathrm{DE}}(t) + (-163 \cdot 10^{-9}) \cdot N_{\mathrm{LU}}(t) + (6 \cdot 10^{-9}) \cdot N_{\mathrm{NL}}(t) + (18 \cdot 10^{-9}) \cdot N_{\mathrm{PL}}(t))$$

Choose a country — Please select a country from the list below: DE - Germany

Choose a date: — Input a date: 2020-03-25

Choose either cumulative or daily mode: — Choose plotting mode: [✓ Cumulative] Daily

Choose a duration for the forecast: — Please select a duration in days from the list below: 5

CASES IN GERMANY

Cumulated and newly incurred cases, monitored at daily frequency from 31/12/2019 until 26/03/2020

CUMULATED CASES BY COUNTRY AS OF 2020-03-25

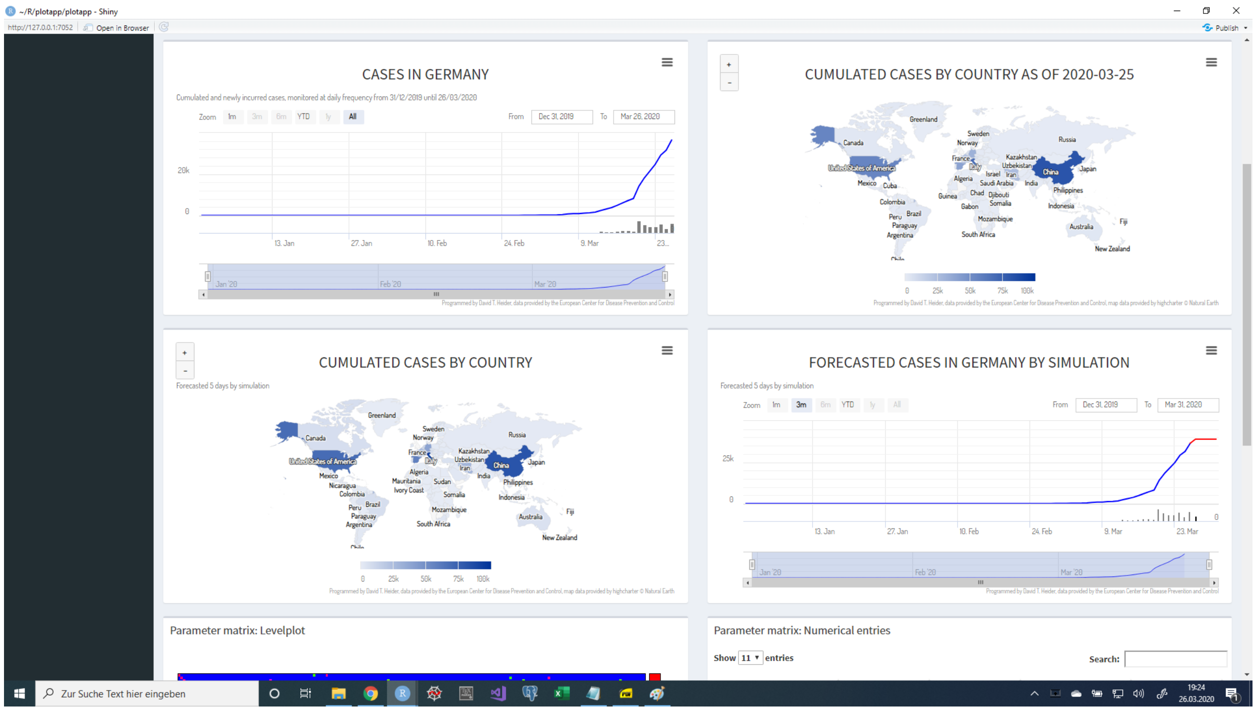
CUMULATED CASES BY COUNTRY

Forecasted 5 days by simulation

FORECASTED CASES IN GERMANY BY SIMULATION

~/R/plotapp/plotapp - Shiny
Intro
Cases
Deaths
Zoom 1m 3m 6m YTD 1y All
From Dec 31, 2019 To Mar 26, 2020
Zur Suche Text hier eingeben
19:23
26.03.2020

Figure 4: Simulation tab - Equation with fitted for cumulated cases for a given and top row: Example for cumulated CoViD-cases in Germany


CASES IN GERMANY

Cumulated and newly incurred cases, monitored at daily frequency from 31/12/2019 until 26/03/2020

CUMULATED CASES BY COUNTRY AS OF 2020-03-25

CUMULATED CASES BY COUNTRY

Forecasted 5 days by simulation

FORECASTED CASES IN GERMANY BY SIMULATION

Forecasted 5 days by simulation

Parameter matrix: Levelplot

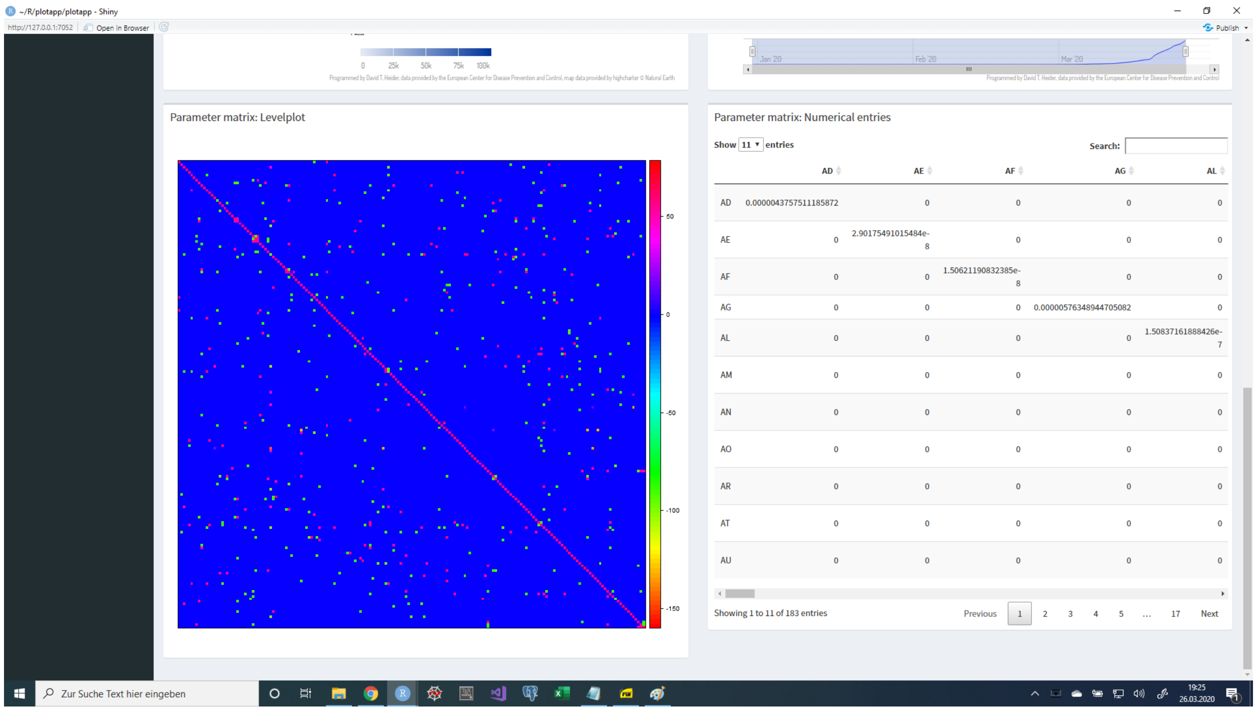
Parameter matrix: Numerical entries

Show 11 entries

Search:

~/R/plotapp/plotapp - Shiny
Zoom 1m 3m 6m YTD 1y All
From Dec 31, 2019 To Mar 31, 2020
Zur Suche Text hier eingeben
19:24
26.03.2020

Figure 5: Simulation tab - Comparison of data (first row) and simulations (second row): Time series examplified for the cumulated CoViD-cases in Germany


~/R/plotapp/plotapp - Shiny
http://127.0.0.1:7052
Open in Browser
Publish
0
25k
50k
75k
100k
Jan '20
Feb '20
Mar '20
Programmed by David T. Heider, data provided by the European Center for Disease Prevention and Control, map data provided by highcharter © Natural Earth
Programmed by David T. Heider, data provided by the European Center for Disease Prevention and Control
Parameter matrix: Levelplot
50
0
-50
-100
-150
Zur Suche Text hier eingeben
19:25
26.03.2020

Parameter matrix: Numerical entries

Show 11 entries Search:

	AD	AE	AF	AG	AL
AD	0.0000043757511185872	0	0	0	0
AE	0	2.90175491015484e-8	0	0	0
AF	0	0	1.50621190832385e-8	0	0
AG	0	0	0	0.00000576348944705082	0
AL	0	0	0	0	1.50837161888426e-7
AM	0	0	0	0	0
AN	0	0	0	0	0
AO	0	0	0	0	0
AR	0	0	0	0	0
AT	0	0	0	0	0
AU	0	0	0	0	0

Showing 1 to 11 of 183 entries Previous 1 2 3 4 5 … 17 Next

Figure 6: Simulation tab - Levelplot of the parameter matrix for the simulation of cumulated cases as well as display of its values in a data grid

```
library(readxl)
library(tidyverse)
library(lubridate)
library(crayon)
library(xts)
library(highcharter)
library(shiny)
library(shinydashboard)
library(readr)
library(stats)
library(RColorBrewer)
library(SimDesign)
library(jsonlite)
library(lattice)
library(DT)

### Auxiliary functions

# Introduction

intro.string.1 <- "This dashboard presents the results of a simple population dynamics transferred to
epidemiology, more precisely, to the study of the spread of the CoViD-\\(19\\) virus over the globe.
Let \\(n\\in\\mathbb{N}\\) denote the number of countries available in (I) the CoViD-dataset provided by
ECDC, (II) the population dataset available as a JSON on github and (III) the countries listed in the
geodatasource dataset that provides pairs of geographically neighboring countries. We consider
\\(\\mathbf{N}=\\sum_{i=1}^n N_i\\hat{e}_i\\in\\left(\\mathbb{R}_0^+\\right)^n\\) as a representation
of the number of people \\(N_i = N_i(t)\\) in country \\(i\\) that have been infected with the virus
```

```
up to time \\(t\\geq 0\\), measured in days. Note that they might have recovered or died at time
\\(T\\leq t\\), yet are counted.<br/><br/>"

intro.string.2 <- "The model we investigate is
$$\\frac{dN_{i}}{dt} = \\left(P_i - N_i(t)\\right)\\cdot\\langle\\mathbf{k}_i,\\mathbf{N}\\rangle,$$
where \\(P_i\\) indicates the respective country's population at time \\(t=0\\). This is a generalizati
of the logistic growth model to include interactions in the vectorial constants.
The \\(\\mathbf{k}_i\\in\\mathbb{R}^n\\), \\(1\\leq i\\leq n\\), can be bound by columns
so as to give rise to a matrix of parameters, \\(K\\in\\mathbb{R}^{n^2}\\).<br/><br/>"

intro.string.3 <- "Practically, the parameter matrix is determined by setting
\\(K_{ij} = \\mathbf{k}_i\\cdot\\hat{e}_j:= 0\\) whenever country \\(i\\) and country \\(j\\) do
not share a geographic border according to the dataset (III). Otherwise, \\(K_{ij}\\) is determined
via linear regression for each \\(i,\\) \\(1\\leq i\\leq n\\), individually to the epidemiological
data provided by the ECDC. Numerically, \\(d_t N_i(t)\\) is identified with the number of new CoViD-case
in the country at a given time \\(t\\). The latter is measured in units of days from \\(2019\\)-
\\(12\\)-\\(31\\) onwards. \\(N_i(t)\\) is obtained from the data by taking the cumulative sum of
the new CoViD-cases up to time \\(t\\), inclusively. For details see the source code below. Since
the regression model is linear in the matrix \\(K\\), the estimate obtained that way is unbiased.
The consistency could not be assessed thoroughly because the number of available datapoints was too
small.<br/><br/>"

intro.string.4 <- "Ultimately, a time-forward simulation is conducted via the finite-differences
method for systems of ordinary differential equations. The method is expounded in the usual
suspects for textbooks on numerical mathematics. The empirical values of \\(N_i(T)\\) serve as
initial condition and \\(dt\\) is approximated by a time-interval of \\(1\\) day, as in the
original data.<br/><br/>"

intro.string.5 <- "The model has been fitted for both reported quantities, the reported CoViD-\\(19\\)
cases and deaths. The results are shown graphically together with the fitting parameters in tab \\(2\\)
and \\(3\\) of the dashboard. The figure in the bottom left corner of tab \\(1\\) depicts a binary view
on the adjacency matrix which indicates in blue if two countries are neighboring. The white pixels indi
that the pair of countries listed as column and axis label share no geographic border in the sense of
dataset (III).<br/><br/>"

intro.string.6 <- "The top row in tabs \\(1\\) and \\(2\\) depict the original data as a time series
for a given country and the spatial distribution at a given point in time for which data has been
gathered by the ECDC and made public. The middle row depicts the spatial forecast \\(\\Delta t\\)
days into the future, starting from the latest observation at \\(t = T\\) as well as a time series
plot for the empirical data (blue) and the simulated data (red) from \\(t = T + 1\\) to \\(t = T + \\Del
i.e,, the forecast horizon. The bottom row depicts the entries of the parameter matrix
\\(K = (K_{ij})_{1\\leq i, j\\leq n}\\) in the form of a scaled level plot. The matrix has been
transformed by applying the mapping $$\\Pi:\\mathbb{R}^2\\to\\mathbb{R}^2,\\, K_{ij}\\mapsto\\log
\\left\\vert\\frac{K_{i,j}}{\\min_{(i,j)}\\lbrace K_{ij}\\vert K_{ij}\\neq 0\\rbrace}\\right\\vert$$
whenever \\(K_{ij}\\neq 0\\). If \\(K_{ij}=0\\), a null-continuation of \\(\\Pi\\) has been used, i.e,
\\(0\\mapsto 0\\). Finally, the data grid in the bottom right corner depicts the numerical values of
the parameter matrix \\(K\\) used in the simulation visualized in the middle row.<br/><br/>"

intro.string.7 <- "Based on the results of this project, the model's predictions should be treated as
possibly reliable for a forecast horizon of at maximum \\(14\\) days from the date of the latest
observation dataset (I). Obvious limitations on the number of available datapoints prohibit further
inference on the dynamics of the virus pandemy in the context of the model. For a considerably short
forecast hoirzon of less than \\(7\\) days, the forecast seems plausible.<br/><br/>"
```

```
intro.string.8 <- "The model as well as the source code is open for the general public for adaptation
and usage as well as development at own responsibility.<br/><br/>"

intro.string.9 <- "<i>David Heider, Munich on 2020-03-26</i>"

# Auxiliary function to retrieve the current CoVidD-data from the ECDC API

read_CoViD_data <- function() {
  tryCatch(
    expr = {
      tf <- paste0(getwd(), "/coviddata_", format(Sys.time(), "%Y-%m-%d"), ".xlsx")
      if(file.exists(tf) == FALSE) {

        url <- paste("https://www.ecdc.europa.eu/sites/default/files/documents/COVID-19-geographic-disbt
                     format(Sys.time(), "%Y-%m-%d"),
                     ".xlsx", sep = "")
        suppressWarnings(download.file(url, tf, mode = "wb", quiet = TRUE))

        cat(red("LATEST COPY NOT AVAILABLE ON SYSTEM.\n"))
        cat(red("DOWNLOAD LATEST COPY!\n"))

      }
      data <- read_xlsx(tf)
      cat(red("1. IMPORTED COVID-19 DATA SUCCESSFULLY!\n"))
      return(data)
    },
    error = function(e) {
      tf <- paste0(getwd(), "/coviddata_", format(Sys.time() - ddays(1), "%Y-%m-%d"), ".xlsx")
      if(file.exists(tf) == FALSE) {

        cat(red("LATEST COPY NOT AVAILABLE ON SYSTEM.\n"))
        cat(red("DOWNLOAD LATEST COPY!\n"))

        url <- paste("https://www.ecdc.europa.eu/sites/default/files/documents/COVID-19-geographic-disbt
                     format(Sys.time() - ddays(1), "%Y-%m-%d"),
                     ".xlsx", sep = "")
        suppressWarnings(download.file(url, tf, mode = "wb", quiet = TRUE))
      }
      data <- read_xlsx(tf)
      cat(red("1. IMPORTED COVID-19 DATA SUCCESSFULLY!\n"))
      return(data)
    })
}

global.covid <- read_CoViD_data()

# Auxiliary function to retrieve border data

read_border_data <- function() {

  tf <- paste0(getwd(), "/coviddata_geodatasource_country_borders", ".csv")

  if(!file.exists(tf)) {
```

```r
    url <- "https://raw.github.com/geodatasource/country-borders/master/GEODATASOURCE-COUNTRY-BORDERS.CS
    suppressWarnings(download.file(url, tf, mode = "wb", quiet = TRUE))
  }
  dat <- quiet(read_csv(tf))

  return(dat)
}

global.border <- read_border_data()

# Auxiliary function to retrieve population data

read_population_data <- function() {
  tf <- paste0(getwd(), "/country_populations", ".json")

  if(!file.exists(tf)) {
    url <- "https://gist.githubusercontent.com/gwillem/6ca8a81048e6f3721c3bafc803d44a72/raw/4fb66d18178
    suppressWarnings(download.file(url, tf, mode = "wb", quiet = TRUE))
  }

  population <- quiet(fromJSON(tf)) %>%
    as.data.frame(stringsAsFactors = FALSE) %>%
    t()
  return(population)
}

global.population <- read_population_data()

# Auxiliary function to tidy CoViD-19 data (not minimal, allows extensions of the app)

tidy_CoViD_data <- function(data.in = read_CoViD_data()) {
  data <- data.in %>%
    data.frame() %>%
    select("DateRep", "Cases", "Deaths", "GeoId") %>%
    filter(GeoId != "JPG11668")

  data.Cases <- data %>%
    select(-Deaths) %>%
    spread(key = GeoId, value = Cases)
  rownames(data.Cases) <- data.Cases[, 1]
  data.Cases <- data.Cases[, -1]
  colnames(data.Cases) <- str_c("Cases.", colnames(data.Cases))
  data.Cases[is.na(data.Cases)] <- "0"
  data.Cases[] <- lapply(data.Cases, function(x) as.numeric(as.character(x)))

  data.CumCases <- apply(data.Cases, 2, cumsum) %>%
    as.data.frame()
  colnames(data.CumCases) <- str_c("Cum", colnames(data.Cases))

  data.ComCases <- merge(data.Cases, data.CumCases, by = 0)
  rownames(data.ComCases) <- data.ComCases[,"Row.names"]
  data.ComCases <- data.ComCases[,-1]
  data.Deaths <- data %>%
    select(-Cases) %>%
    spread(key = GeoId, value = Deaths)
  rownames(data.Deaths) <- data.Deaths[, 1]
  data.Deaths <- data.Deaths[, -1]
  colnames(data.Deaths) <- str_c("Deaths.", colnames(data.Deaths))
  data.Deaths[is.na(data.Deaths)] <- "0"
  data.Deaths[] <- lapply(data.Deaths, function(x) as.numeric(as.character(x)))

  data.CumDeaths <- apply(data.Deaths, 2, cumsum) %>%
    as.data.frame()
  colnames(data.CumDeaths) <- str_c("Cum", colnames(data.Deaths))

  data.ComDeaths <- merge(data.Deaths, data.CumDeaths, by = 0)
  rownames(data.ComDeaths) <- data.ComDeaths[,"Row.names"]
  data.ComDeaths <- data.ComDeaths[,-1]

  data <- merge(data.ComCases, data.ComDeaths, by = 0)
  rownames(data) <- data[,"Row.names"]
  colnames(data)[1] <- "Date"
  data[,1] <- data[,1] %>%
    as.character() %>%
    as_date() %>%
    as.POSIXct()
  data <- xts(data[,-1], order.by = data[,1])
  cat(red("2. TIDIED COVID-19 DATA SUCCESSFULLY!\n"))
  return(data)
}

# Auxiliary function to obtain a renderable highcharter object for CoViD-19 cases

plot_CoViD_cases <- function(country.id, data = read_CoViD_data()) {
  diagram.data <- data %>%
    tidy_CoViD_data()

  country.name <-  data %>%
    select("GeoId", "Countries and territories") %>%
    unique() %>%
    filter(GeoId == country.id) %>%
    select("Countries and territories") %>%
    pull() %>%
    str_replace_all(pattern = "_", replacement = " ")

  title.text <- paste0("<b><h3>Cases in ", country.name, " </h3></b>")
  cum.cases <- str2lang(paste0("diagram.data$CumCases.", country.id)) %>%
    eval()
  cum.text <- paste("Cumulated cases in", country.name, sep = " ")
  new.cases <- str2lang(paste0("diagram.data$Cases.", country.id)) %>%
    eval()
  new.text <- paste("Daily new cases in", country.name, sep = " ")

  cat(red("3. GATHERED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))
  hc <- highchart(type = "stock") %>%
    hc_yAxis_multiples(create_yaxis(2, height = c(5, 1), turnopposite = TRUE)) %>%
    hc_title(text = title.text, useHTML = TRUE, align = "center") %>%
    hc_subtitle(text = paste0("Cumulated and newly incurred cases, monitored at daily frequency from 31/
                              format(Sys.time(), "%d%/%m/%Y")),
                align = "left") %>%
    hc_add_series(cum.cases,
                  yAxis = 0,
                  type = "line",
                  name = cum.text,
                  color = "blue") %>%
    hc_add_series(new.cases,
                  yAxis = 1,
                  type = "column",
                  color = "gray",
                  name = new.text) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
               text = "Programmed by David T. Heider, data provided by the European Center for Disease 
               href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-
    hc_add_theme(hc_theme_gridlight())

  cat(red("4. PLOTTED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))
  return(hc)
}

# Auxiliary function to obtain a renderable highcharter object for CoViD-19 deaths

plot_CoViD_deaths <- function(country.id, data = read_CoViD_data()) {
  diagram.data <- data %>%
    tidy_CoViD_data()

  country.name <-  data %>%
    select("GeoId", "Countries and territories") %>%
    unique() %>%
    filter(GeoId == country.id) %>%
    select("Countries and territories") %>%
    pull() %>%
    str_replace_all(pattern = "_", replacement = " ")

  title.text <- paste0("<b><h3>Deaths in ", country.name, " </h3></b>")
  cum.cases <- str2lang(paste0("diagram.data$CumDeaths.", country.id)) %>%
    eval()
  cum.text <- paste("Cumulated deaths in", country.name, sep = " ")
  new.cases <- str2lang(paste0("diagram.data$Deaths.", country.id)) %>%
    eval()
  new.text <- paste("Daily new deaths in", country.name, sep = " ")

  cat(red("3. GATHERED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))

  hc <- highchart(type = "stock") %>%
    hc_yAxis_multiples(create_yaxis(2, height = c(5, 1), turnopposite = TRUE)) %>%
    hc_title(text = title.text, useHTML = TRUE, align = "center") %>%
```

```r
    hc_subtitle(text = paste0("Cumulated and newly incurred deaths, monitored at daily frequency from 3
                              format(Sys.time(), "%d%/%m/%Y")),
                align = "left") %>%
    hc_add_series(cum.cases,
                  yAxis = 0,
                  type = "line",
                  name = cum.text,
                  color = "blue") %>%
    hc_add_series(new.cases,
                  yAxis = 1,
                  type = "column",
                  color = "gray",
                  name = new.text) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
               text = "Programmed by David T. Heider, data provided by the European Center for Disease 
               href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-
    hc_add_theme(hc_theme_gridlight())

  cat(red("4. PLOTTED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))
  return(hc)
}

# Auxiliary function to retrieve a list of country names and GeoIDs

get_countries_id <- function() {
  data <- global.covid %>%
    select("Countries and territories", "GeoId") %>%
    unique()
  colnames(data) <- c("country", "country.id")
  data <- data %>%
    mutate_at(.vars = c("country", "country.id"),
              .funs = function(x) str_replace_all(x, pattern = "_", replacement = " "))
  data <- data %>%
    filter(country.id != "JPG11668") %>%
    as.data.frame()
  return(data)
}

# Auxiliary function to concatenate GeoID in Iso-A2 format and country name.

get_display_id <- function() {
  country.names <- get_countries_id() %>%
    select("country") %>%
    pull()
  country.ids <- get_countries_id() %>%
    select("country.id") %>%
    pull()
  id.list <- map2_chr(.x = country.ids,
                      .y = country.names,
                      .f = function(x, y) {
                        return(str_c(x, y, sep = " - "))
                      })
  return(id.list)
}

# Auxiliary function to set the minimum allowed date for plotting

get_min_date <- function() {
  min.date <- global.covid %>%
    select("DateRep") %>%
    unique() %>%
    summarize(min.date = min(DateRep)) %>%
    pull(min.date)
  return(min.date)
}

# Auxiliary function to set the maximum allowed date for plotting

get_max_date <- function() {
  max.date <- global.covid %>%
    select("DateRep") %>%
    unique() %>%
    summarize(max.date = max(DateRep)) %>%
    pull(max.date)
  return(max.date)
}

# Auxiliary function to plot chloropleth

geo_plot_CoViD_data <- function(date.id = "2020-03-21", selected.value = "CumCases") {
  date.id <- as.character(date.id)
  selected.value <- as.character(selected.value)

  selected.value.longlist <- c("Cumulated cases", "Cumulated deaths", "Today's new cases", "Today's new
  names(selected.value.longlist) <- c("CumCases", "CumDeaths", "Cases", "Deaths")
  selected.value.long <- selected.value.longlist[[selected.value]]

  geo.data <- global.covid %>%
    select("DateRep", "GeoId", "Cases", "Deaths") %>%
    rename(country.id = GeoId, time.id = DateRep) %>%
    filter(country.id != "JPG11668") %>%
    arrange(time.id) %>%
    group_by(country.id) %>%
    mutate(CumCases = cumsum(Cases), CumDeaths = cumsum(Deaths)) %>%
    ungroup() %>%
    arrange(desc(time.id)) %>%
    group_by(time.id) %>%
    nest() %>%
    rename(nested.data = data) %>%
    as.data.frame()
  row.names(geo.data) <- geo.data$time.id
  geo.data[,1] <- NULL

  cat(red("5. GATHERED REQUESTED GEO-COVID-19 DATA SUCCESSFULLY!\n"))
  test.data <- geo.data[date.id, "nested.data"][[1]] %>%
    as.data.frame()

  # Set download_map_data = FALSE and set
  # tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js")
  # in the container of the hcOutput.

  hc <- hcmap("custom/world-lowres",
              download_map_data = FALSE,
              data = test.data,
              value = selected.value,
              joinBy = c("iso-a2", "country.id"),
              name = selected.value.long,
              dataLabels = list(enabled = TRUE, format = '{point.name}'),
              borderWidth = 0.3) %>%
    hc_title(text = str_c("<b><h3>", selected.value.long," by country as of ", date.id, "</h3></b>"),
             useHTML = TRUE, align = "center") %>%
    hc_mapNavigation(enabled = TRUE) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
               text = "Programmed by David T. Heider, data provided by the European Center for Disease I
               href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-d
    hc_add_theme(hc_theme_gridlight())

  cat(red("6. PLOTTED REQUESTED GEO-COVID-19 DATA SUCCESSFULLY!\n"))

  return(hc)

}

# Auxiliary function to calculate the adjacency matrix for countries in the epidemiological model.
# NB: This function takes several seconds to execute on a standard system, thus we will import the resu

calculate_CoViD_adjacency_matrix <- function() {

  cat(red("7. Calculating adjacency matrix. This may take some time...\n"))

  dat <- global.border %>%
    select(country_code, country_border_code) %>%
    mutate(country_code = as.character(country_code), country_border_code = as.character(country_border_

  codes.covid <- global.covid %>%
    select(GeoId) %>%
    mutate(GeoId = as.character(GeoId)) %>%
    filter(GeoId != "JPG11668") %>%
    pull() %>%
    unique() %>%
    as.vector()

  dat <- dat %>%
    filter(country_border_code %in% codes.covid) %>%
    filter(country_code %in% codes.covid)
```

```r
  adjacency.matrix <- diag(length(codes.covid))

  rownames(adjacency.matrix) <- codes.covid
  colnames(adjacency.matrix) <- codes.covid

  for(i in 1:length(codes.covid)) {
    for(j in i:length(codes.covid)) {
      if(j > i) {
        adjacency.matrix[j, i] <- adjacency.matrix[i, j] <- dat %>%
          rbind(data.frame(country_code = codes.covid[i], country_border_code = codes.covid[j])) %>%
          base::duplicated() %>%
          tail(1) %>%
          as.numeric()
      }
      else{
        # Do nothing
      }
    }
  }

  cat(red("8. Finished calculating adjacency matrix!\n"))

  return(adjacency.matrix)

}

fn.adj <- paste0(getwd(), "/adjacency.Rdata")

if(file.exists(fn.adj)) {
  load(fn.adj)
} else {
  adjacency.matrix <- calculate_CoViD_adjacency_matrix()
  save(adjacency.matrix, file = fn.adj)
}

# Get boolean mask from adjacency matrix

adjacency.matrix.lgl <- ifelse(adjacency.matrix == 1, TRUE, FALSE)

# Auxiliary function to massage data in a way so that we can estimate the parameters of 1 model...

prepare_glm_CoViD <- function(country.selection = "DE", selected.value = "Cases") {

  country.selection <- as.character(country.selection)
  selected.value <- as.character(selected.value)

  population <- global.population

  codes.population <- rownames(population)

  codes.covid <- global.covid %>%
    select(GeoId) %>%
    mutate(GeoId = as.character(GeoId)) %>%
    filter(GeoId != "JPG11668") %>%
    pull() %>%
    unique() %>%
    as.vector()

  codes.reduced <- codes.population[codes.population %in% codes.covid]

  population.reduced <- population[codes.reduced,] %>% as.numeric()
  names(population.reduced) <- codes.reduced

  adjacency.matrix.lgl.reduced <- adjacency.matrix.lgl[codes.reduced %in% rownames(adjacency.matrix.lgl)

  pop <- population.reduced[country.selection]

  temp.stuff <- names(adjacency.matrix.lgl.reduced[,country.selection])
  temp.stuff <- temp.stuff[adjacency.matrix.lgl.reduced[,country.selection]]

  int.codes <- codes.reduced[codes.reduced %in% temp.stuff]

  if(length(int.codes) > 1) {
    geo.data.1 <- global.covid %>%
      select(time.id = DateRep, country.id = GeoId, all_of(selected.value)) %>%
      filter(country.id != "JPG11668") %>%
      filter(country.id %in% int.codes) %>%
      arrange(time.id)  %>%
      spread(key = country.id, value = selected.value, sep = ".")

    geo.data.1[is.na(geo.data.1)] <- 0

    geo.data.2 <- geo.data.1 %>%
      gather("country.id", selected.value, -time.id) %>%
      mutate(country.id = substr(country.id, nchar(country.id)-1, nchar(country.id))) %>%
      group_by(country.id) %>%
      mutate(cumul.value = cumsum(selected.value)) %>%
      ungroup() %>%
      select(-selected.value) %>%
      spread(key = country.id, value = cumul.value, sep = ".")

    geo.names1 <- colnames(geo.data.1)[-1] %>%
      map_chr(.f = function(x) {
        return(str_replace(x, "country.id", "diff"))
      })
    geo.data.1 <- geo.data.1[,-1]
    colnames(geo.data.1) <- geo.names1

    geo.data.1 <- geo.data.1[paste0("diff.", country.selection)]

    geo.names2 <- colnames(geo.data.2)[-1] %>%
      map_chr(.f = function(x) {
        return(str_replace(x, "country.id", "cum"))
      })
    geo.data.2 <- geo.data.2[,-1]
    colnames(geo.data.2) <- geo.names2
```

```
    geo.data.3 <- pop - geo.data.2[paste0("cum.", country.selection)]
    colnames(geo.data.3) <- paste0("popmcum.", country.selection)

    geo.data.2 <- t(apply(geo.data.2,
                          1,
                          function(x1) {
                            return(
                              map2_dbl(.x = x1,
                                       .y = seq_along(x1),
                                       .f = function(a, b) {
                                         return(a * geo.data.3[b, 1])
                                       }
                              )
                            )
                          }
    )) %>%
      as.data.frame()

    colnames(geo.data.2) <- colnames(geo.data.2) %>%
      map_chr(.f = function(x) {
        return(str_replace(x, "cum", "var"))
      })

    geo.data.complete <- geo.data.1 %>%
      cbind.data.frame(geo.data.2)
  } else if(length(int.codes) == 1) {
    geo.data.1 <- global.covid %>%
      select(time.id = DateRep, country.id = GeoId, all_of(selected.value)) %>%
      filter(country.id != "JPG11668") %>%
      filter(country.id %in% int.codes) %>%
      arrange(time.id) %>%
      spread(key = country.id, value = selected.value, sep = ".", drop = FALSE)

    geo.data.1[is.na(geo.data.1)] <- 0

    geo.data.2 <- geo.data.1 %>%
      gather("country.id", selected.value, -time.id) %>%
      mutate(country.id = substr(country.id, nchar(country.id)-1, nchar(country.id))) %>%
      mutate(cumul.value = cumsum(selected.value)) %>%
      select(-selected.value, - country.id)

    geo.names1 <- colnames(geo.data.1)[-1] %>%
      map_chr(.f = function(x) {
        return(str_replace(x, "country.id", "diff"))
      })
    geo.data.1 <- geo.data.1[,-1]
    colnames(geo.data.1) <- geo.names1

    geo.data.1 <- geo.data.1[paste0("diff.", country.selection)]

    geo.names2 <- colnames(geo.data.2)[-1] %>%
      str_replace("cumul.value", paste0("cum.", country.selection))
    geo.data.2 <- geo.data.2[,-1]
    colnames(geo.data.2) <- geo.names2

    geo.data.3 <- pop - geo.data.2[paste0("cum.", country.selection)]
    colnames(geo.data.3) <- paste0("popmcum.", country.selection)

    geo.data.2 <- apply(geo.data.2,
                        1,
                        function(x1) {
                          return(
                            map2_dbl(.x = x1,
                                     .y = seq_along(x1),
                                     .f = function(a, b) {
                                       return(a * geo.data.3[b, 1])
                                     }
                            )
                          )
                        }
    ) %>%
      as.data.frame()

    colnames(geo.data.2) <- paste0("var.", country.selection)

    geo.data.complete <- geo.data.1 %>%
      cbind.data.frame(geo.data.2)
  } else{
    # Do nothing
  }

  return(geo.data.complete)
}

# Auxiliary function to calculate the parameters for one target country and one mode

get_glm_CoViD <- function(country.selection = "DE", selected.value = "Cases") {

  country.selection <- as.character(country.selection)
  selected.value <- as.character(selected.value)

  input.data <- prepare_glm_CoViD(country.selection = country.selection, selected.value = selected.valu

  input.data.temp <- input.data[rowSums(input.data) > 0, ]
  if(length(input.data.temp) > 0) {
    input.data <- input.data.temp
  }

  measurevar <- colnames(input.data)[1]
  groupvars <- colnames(input.data)[-1]

  string.formula <- paste(measurevar, paste(groupvars, collapse = " + "), sep = " ~ ")
  string.formula <- paste(string.formula, "0", sep = " +") # no whitespace here!
  formula.final <- as.formula(string.formula)
  tryCatch(
    expr = {
      param.model <- lm(formula = formula.final,
                        data = input.data)
      out.coeffs <- param.model$coefficients

      names(out.coeffs) <- str_replace(names(out.coeffs), pattern = "var.", replacement = paste0(countr

      return(out.coeffs)
    },
    error = function(e) {
      out.coeffs <- rep(0, length(groupvars))
      names(out.coeffs) <- groupvars

      names(out.coeffs) <- str_replace(names(out.coeffs), pattern = "var.", replacement = paste0(countr

      return(out.coeffs)
  })
}

# Auxiliary function to obtain the (interaction)-model parameters.

get_int_model_params <- function(selected.mode = "Cases") {

  selected.mode <- as.character(selected.mode)

  population <- global.population

  codes.population <- rownames(population)

  codes.covid <- global.covid %>%
    select(GeoId) %>%
    mutate(GeoId = as.character(GeoId)) %>%
    filter(GeoId != "JPG11668") %>%
    pull() %>%
    unique() %>%
    as.vector()

  codes.reduced <- codes.population[codes.population %in% codes.covid]

  adjacency.matrix.lgl.reduced <- adjacency.matrix.lgl[codes.reduced %in% rownames(adjacency.matrix.lgl

  temp.stuff <- rownames(adjacency.matrix.lgl.reduced)

  codes.reduced <- codes.reduced[codes.reduced %in% temp.stuff]

  result <- c()

  for(i in 1:length(codes.reduced)) {
    tryCatch(
      expr = {
        temp.result <- get_glm_CoViD(country.selection = codes.reduced[i], selected.value = selected.mo
        cat(green(paste0("Processed country with ISO-2-letter code: ", codes.reduced[i], "!\n")))
      result <- c(result, temp.result)
      },
      error = function(e) {
        cat(red(paste0("Error while processing country with ISO-2-letter code: ", codes.reduced[i], "!\
        cat(red("Wait for 1 second.\n"))
        Sys.sleep(1)
      }
    )
  }

  result <- result[!is.na(result)]

  cat(red("\nOBTAINED MODEL PARAMETERS SUCCESSFULLY!\n"))

  return(result)
}

# The data from the following command will be loaded:
# cases.params <- get_int_model_params(selected.mode = "Cases")

fn.cases <- paste0(getwd(), "/cases_params_", format(Sys.time(), "%Y-%m-%d"), ".Rdata")

if(file.exists(fn.cases)) {
  load(fn.cases)
} else {
  cases.params <- get_int_model_params(selected.mode = "Cases")
  save(cases.params, file = fn.cases)
}

# Output: cases.params

# The data from the following command will be loaded:
# deaths.params <- get_int_model_params(selected.mode = "Deaths")

fn.deaths <- paste0(getwd(), "/deaths_params_", format(Sys.time(), "%Y-%m-%d"), ".Rdata")

if(file.exists(fn.deaths)) {
  load(fn.deaths)
} else {
  deaths.params <- get_int_model_params(selected.mode = "Deaths")
  save(deaths.params, file = fn.deaths)
}

# Auxiliary function to convert the named numerical vector into a matrix
# Row variables: Target country
# Column variables: Countries which influence the CoViD spread according to the model.
# Structure makes matrix algebra possible.

produce_CoViD_param_matrix <- function(selected.mode = "Cases")
{
  if(selected.mode == "Cases") {
    dimensions <- names(cases.params) %>%
      substr(start = 1L, stop = 2L) %>%
      unique() # This gives a list of countries with CoVid, population and interaction data available
    param.matrix <- diag(nrow = length(dimensions), names = FALSE)
    rownames(param.matrix) <- dimensions
    colnames(param.matrix) <- dimensions
    for(i in seq_along(dimensions)) {
      for(j in seq_along(dimensions)) {
        param.matrix[i, j] <- as.numeric(cases.params[paste0(dimensions[i], ".by.", dimensions[j])])
      }
    }
    param.matrix[is.na(param.matrix)] <- 0
    return(param.matrix)
  } else if(selected.mode == "Deaths") {
    dimensions <- names(deaths.params) %>%
      substr(start = 1L, stop = 2L) %>%
      unique() # This gives a list of countries with CoVid, population and interaction data available
    param.matrix <- diag(nrow = length(dimensions), names = FALSE)
    rownames(param.matrix) <- dimensions
    colnames(param.matrix) <- dimensions
    for(i in seq_along(dimensions)) {
      for(j in seq_along(dimensions)) {
        param.matrix[i, j] <- as.numeric(deaths.params[paste0(dimensions[i], ".by.", dimensions[j])])
      }
    }
    param.matrix[is.na(param.matrix)] <- 0
    return(param.matrix)
  } else {
    produce_CoViD_param_matrix(selected.mode = "Cases")
  }
}

# Auxiliary function to plot parameter matrix

plot_CoViD_param_matrix <- function(selected.mode = "Cases") {

  selected.mode <- as.character(selected.mode)
  param.matrix <- produce_CoViD_param_matrix(selected.mode = selected.mode)

  pmt <- ifelse(param.matrix != 0,
                sign(param.matrix)*(-log(min(abs(param.matrix[param.matrix != 0]))))+log(abs(param.matri
                0)

  output <- levelplot(t(pmt[c(nrow(pmt):1),]),
                      xlab = NULL,
                      ylab = NULL,
                      col.regions=rainbow(100, rev = FALSE),
                      labels = FALSE,
                      cuts = 100,
                      scales = list(draw = FALSE))
  return(output)

}

# Auxiliary function to get the ISO-2-letter code associated to the parameter matrix of the model for e
```

```r
get_model_names <- function(selected.mode = "Cases") {
  if(selected.mode == "Cases") {
    return(rownames(produce_CoViD_param_matrix(selected.mode = "Cases"))) } else {
      return(rownames(produce_CoViD_param_matrix(selected.mode = "Deaths")))
    }
}

# Auxiliary function to produce most of ISO-2 - NLP pairs to display on first page.

get_name_dictionary <- function(selected.mode = "Cases") {

  selected.mode <- as.character(selected.mode)

  dat <- global.border %>%
    select(ID = country_code, Name = country_name) %>%
    filter(ID %in% get_model_names(selected.mode = selected.mode)) %>%
    unique()
  return(dat)
}

# Auxiliary function to extract cumulated cases of or cumulated deaths due to CoViD-19 from ECDC

get_initial_cumvalue <- function(selected.mode = "Cases") {

  selected.mode <- as.character(selected.mode)

  relevant.countries <- get_model_names(selected.mode = selected.mode)

  initial.values <- global.covid %>%
    select("GeoId", "Values" = all_of(selected.mode)) %>%
    filter(GeoId %in% relevant.countries) %>%
    group_by(GeoId) %>%
    summarise(cumvalue = sum(Values, na.rm = TRUE))

  result <- rep(0, length(relevant.countries))
  names(result) <- relevant.countries
  result[initial.values$GeoId] <- initial.values$cumvalue

  return(result)
}

# Auxiliary function to extract populations from a GitHub JSON.
# Note: As of 2020-03-25 (to my awareness), the ECDC data contains population info as of 2018.
# As I have calibrated the model with another dataset, I will continue to use this one further for the

get_initial_population <- function(selected.mode = "Cases") {

  selected.mode <- as.character(selected.mode)

  relevant.countries <- get_model_names(selected.mode = selected.mode)

  tf <- paste0(getwd(), "/country_populations", ".json")
  if(!file.exists(tf)) {
    url <- "https://gist.githubusercontent.com/gwillem/6ca8a81048e6f3721c3bafc803d44a72/raw/4fb66d18178
    suppressWarnings(download.file(url, tf, mode = "wb", quiet = TRUE))
  }
  population <- quiet(fromJSON(tf)) %>%
    as.data.frame(stringsAsFactors = FALSE)

  result <- population[,relevant.countries] %>%
    as.vector() %>%
    as.numeric()

  names(result) <- relevant.countries

  return(result)
}

# Auxiliary function that performs the simulation

do_simulation <- function(selected.mode = "Cases", duration = 14) {

  selected.mode <- as.character(selected.mode)
  duration <- as.integer(duration)

  params <- produce_CoViD_param_matrix(selected.mode = selected.mode)
  initial.pop <- get_initial_population(selected.mode = selected.mode)
  initial.cumvalue <- get_initial_cumvalue(selected.mode = selected.mode)

  start.date <- get_max_date()
  dates <- start.date + ddays(x = 0:duration)

  simulated <- initial.cumvalue
  simulated.matrix <- matrix(0, nrow = 1 + duration, ncol = length(simulated))
  simulated.matrix[1,] <- simulated
  colnames(simulated.matrix) = names(simulated)

  for(i in 1:duration){

    # Save previous result
    previous <- simulated

    # Step forward
    simulated <- as.vector(simulated + (initial.pop - simulated) * (params %*% simulated))

    # Cut off if exceed minimum allowed values
    simulated[simulated < previous] <- previous[simulated < previous]
    simulated[simulated == -Inf] <- previous[simulated < previous]

    # Cut off if exceed maximum allowed values
    simulated[simulated == Inf] <- initial.pop[simulated == Inf]
    simulated[simulated > initial.pop] <- initial.pop[simulated > initial.pop]

    # Round and assign back
    simulated.matrix[i+1,] <- simulated %>%
```

```r
      trunc()
  }

  rownames(simulated.matrix) <- dates

  return(simulated.matrix)
}

# Auxiliary function:
# Does the same simulation as above, but:
# returns an xts object which can be plotted using highcharter

do_simulation_xts <- function(selected.mode = "Cases", duration = 14) {

  selected.mode <- as.character(selected.mode)
  duration <- as.integer(duration)

  params <- produce_CoViD_param_matrix(selected.mode = selected.mode)
  initial.pop <- get_initial_population(selected.mode = selected.mode)
  initial.cumvalue <- get_initial_cumvalue(selected.mode = selected.mode)

  start.date <- get_max_date()
  dates <- start.date + ddays(x = 0:duration)

  simulated <- initial.cumvalue
  simulated.matrix <- matrix(0, nrow = 1 + duration, ncol = length(simulated))
  simulated.matrix[1,] <- simulated
  colnames(simulated.matrix) = names(simulated)

  for(i in 1:duration){

    # Save previous value
    previous <- simulated

    # Step forward
    simulated <- as.vector(simulated + (initial.pop - simulated) * (params %*% simulated))

    # Cut off if exceed minimum allowed values
    simulated[simulated < previous] <- previous[simulated < previous]
    simulated[simulated == -Inf] <- previous[simulated < previous]

    # Cut off if exceed maximum allowed values
    simulated[simulated == Inf] <- initial.pop[simulated == Inf]
    simulated[simulated > initial.pop] <- initial.pop[simulated > initial.pop]

    # Round and assign back
    simulated.matrix[i+1,] <- simulated %>%
      trunc()
  }

  simulated.matrix <- xts(simulated.matrix, order.by = dates)

  return(simulated.matrix)
```

```
}

# Auxiliary function to plot simulation results for cases for a specified duration and country,
# with the duration measured in days from the latest observation.

plot_simulated_CoViD_cases <- function(duration = 14, selected.country = "DE") {

  selected.country <- as.character(selected.country)

  # Prepare historic series

  data <- global.covid

  historic.data <- data %>%
    tidy_CoViD_data()

  country.name <-  data %>%
    select("GeoId", "Countries and territories") %>%
    unique() %>%
    filter(GeoId == selected.country) %>%
    select("Countries and territories") %>%
    pull() %>%
    str_replace_all(pattern = "_", replacement = " ")

  title.text <- paste0("<b><h3>Forecasted cases in ", country.name, " by simulation</h3></b>")

  cum.cases <- str2lang(paste0("historic.data$CumCases.", selected.country)) %>%
    eval()
  cum.text <- paste("Cumulated cases in", country.name, sep = " ")

  new.cases <- str2lang(paste0("historic.data$Cases.", selected.country)) %>%
    eval()
  new.text <- paste("Daily new cases in", country.name, sep = " ")

  cat(red("3. GATHERED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))

  # Prepare simulated series

  cum.simulation.data <- do_simulation_xts(selected.mode = "Cases",
                                          duration = duration)

  cases.simulation.data <- diff.xts(cum.simulation.data)
  cases.simulation.data <-  cases.simulation.data[-1, ]

  cases.sim <- str2lang(paste0("cases.simulation.data$", selected.country)) %>%
    eval()
  cases.sim.text <- paste("Simulated daily new cases in", country.name, sep = " ")

  cum.sim <- str2lang(paste0("cum.simulation.data$", selected.country)) %>%
    eval()
  cum.sim.text <- paste("Simulated cumulated cases in", country.name, sep = " ")

  cat(red("9. Simulated REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))
```

```r
  # Prepare highcharter

  hc <- highchart(type = "stock") %>%
    hc_yAxis_multiples(create_yaxis(2, height = c(5, 1), turnopposite = TRUE)) %>%
    hc_title(text = title.text, useHTML = TRUE, align = "center") %>%
    hc_subtitle(text = paste0("Forecasted ",
                              duration,
                              " days by simulation"),
                align = "left") %>%
    hc_add_series(cum.cases,
                  yAxis = 0,
                  type = "line",
                  name = cum.text,
                  color = "blue") %>%
    hc_add_series(cum.sim,
                  yAxis = 0,
                  type = "line",
                  name = cum.sim.text,
                  color = "red") %>%
    hc_add_series(new.cases,
                  yAxis = 1,
                  type = "column",
                  color = "gray",
                  name = new.text) %>%
    hc_add_series(cases.sim,
                  yAxis = 1,
                  type = "column",
                  color = "black",
                  name = cases.sim.text) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
               text = "Programmed by David T. Heider, data provided by the European Center for Disease I
               href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-d
    hc_add_theme(hc_theme_gridlight())

  cat(red("10. PLOTTED HISTORIC AND SIMULATED COVID-19 DATA SUCCESSFULLY!\n"))
  return(hc)
}

# Auxiliary function to plot simulated deaths for a specified duration in days,
# starting from the latest observation and country

plot_simulated_CoViD_deaths <- function(duration = 14, selected.country = "DE") {

  selected.country <- as.character(selected.country)

  # Prepare historic series

  data <- global.covid

  historic.data <- data %>%
    tidy_CoViD_data()
```

```r
country.name <-  data %>%
  select("GeoId", "Countries and territories") %>%
  unique() %>%
  filter(GeoId == selected.country) %>%
  select("Countries and territories") %>%
  pull() %>%
  str_replace_all(pattern = "_", replacement = " ")

title.text <- paste0("<b><h3>Forecasted deaths in ", country.name, " by simulation</h3></b>")

cum.deaths <- str2lang(paste0("historic.data$CumDeaths.", selected.country)) %>%
  eval()
cum.text <- paste("Cumulated deaths in", country.name, sep = " ")

new.deaths <- str2lang(paste0("historic.data$Deaths.", selected.country)) %>%
  eval()
new.text <- paste("Daily new deaths in", country.name, sep = " ")

cat(red("3. GATHERED REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))

# Prepare simulated series

cum.simulation.data <- do_simulation_xts(selected.mode = "Deaths",
                                         duration = duration)

deaths.simulation.data <- diff.xts(cum.simulation.data)
deaths.simulation.data <-  deaths.simulation.data[-1, ]

deaths.sim <- str2lang(paste0("deaths.simulation.data$", selected.country)) %>%
  eval()
deaths.sim.text <- paste("Simulated daily new deaths in", country.name, sep = " ")

cum.sim <- str2lang(paste0("cum.simulation.data$", selected.country)) %>%
  eval()
cum.sim.text <- paste("Simulated cumulated deaths in", country.name, sep = " ")

cat(red("9. Simulated REQUESTED COVID-19 DATA SUCCESSFULLY!\n"))

# Prepare highcharter

hc <- highchart(type = "stock") %>%
  hc_yAxis_multiples(create_yaxis(2, height = c(5, 1), turnopposite = TRUE)) %>%
  hc_title(text = title.text, useHTML = TRUE, align = "center") %>%
  hc_subtitle(text = paste0("Forecasted ",
                            duration,
                            " days by simulation"),
              align = "left") %>%
  hc_add_series(cum.deaths,
                yAxis = 0,
                type = "line",
                name = cum.text,
                color = "blue") %>%
  hc_add_series(cum.sim,
                   yAxis = 0,
                   type = "line",
                   name = cum.sim.text,
                   color = "red") %>%
    hc_add_series(new.deaths,
                   yAxis = 1,
                   type = "column",
                   color = "gray",
                   name = new.text) %>%
    hc_add_series(deaths.sim,
                   yAxis = 1,
                   type = "column",
                   color = "black",
                   name = deaths.sim.text) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
                text = "Programmed by David T. Heider, data provided by the European Center for Disease 
                href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-
    hc_add_theme(hc_theme_gridlight())

  cat(red("10. PLOTTED HISTORIC AND SIMULATED COVID-19 DATA SUCCESSFULLY!\n"))
  return(hc)
}

geo_plot_simulated_CoViD_data <- function(duration = 14, selected.mode = "Cases") {

  selected.value <- as.character(selected.mode)
  duration <- as.integer(duration)

  selected.value.longlist <- c("Cumulated cases", "Cumulated deaths")
  names(selected.value.longlist) <- c("Cases", "Deaths")
  selected.value.long <- selected.value.longlist[[selected.value]]

  geo.data <- do_simulation(selected.mode = selected.value, duration = duration) %>%
    as.data.frame()
  countries <- colnames(geo.data) %>%
    as.data.frame()

  geo.data <- geo.data[duration, ] %>%
    t()

  geo.data <- cbind(geo.data, countries)
  colnames(geo.data) <- c(selected.value, "country.id")

  cat(red("9. SIMULATED REQUESTED GEO-COVID-19 DATA SUCCESSFULLY!\n"))

  # Set download_map_data = FALSE and set
  # tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js")
  # in the container of the hcOutput.

  hc <- hcmap(map = "custom/world-lowres",
              download_map_data = FALSE,
              data = geo.data,
```

```r
               value = selected.value,
               joinBy = c("iso-a2", "country.id"),
               name = selected.value.long,
               dataLabels = list(enabled = TRUE, format = '{point.name}'),
               borderWidth = 0.3) %>%
    hc_title(text = str_c("<b><h3>", selected.value.long," by country</h3></b>"),
             useHTML = TRUE, align = "center") %>%
    hc_subtitle(text = paste0("Forecasted ",
                              duration,
                              " days by simulation"),
                align = "left") %>%
    hc_mapNavigation(enabled = TRUE) %>%
    hc_exporting(enabled = TRUE) %>%
    hc_credits(enabled = TRUE,
               text = "Programmed by David T. Heider, data provided by the European Center for Disease 
               href = "https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-
    hc_add_theme(hc_theme_gridlight())

  cat(red("10. PLOTTED SIMULATED GEO-COVID-19 DATA SUCCESSFULLY!\n"))

  return(hc)
}

# Auxiliary function to obtain model equation in LaTeX-MathJax-display

get_model_equation <- function(selected.mode = "Cases", selected.country = "DE") {

  selected.mode <- as.character(selected.mode)
  selected.country <- as.character(selected.country)

  country.id <- substr(selected.country, start = 1L, stop = 2L)

  # We loose 12 entries unfortunately.
  tryCatch(
    expr = {
      matrix.scrape <- trunc(produce_CoViD_param_matrix(selected.mode = selected.mode) * 1E9)
      names.scrape <- rownames(matrix.scrape)
      matrix.scrape <- matrix.scrape[country.id, ]

      names.scrape<- names.scrape[matrix.scrape != 0]
      matrix.scrape <- matrix.scrape[matrix.scrape != 0]

      matrix.scrape <- as.character(as.vector(matrix.scrape))

      if(length(matrix.scrape) > 0) {
        string.start <- paste0("$$\\frac{dN_{\\text{",
                               country.id,
                               "}}}{dt}(t) = \\left(P_{\\text{",
                               country.id,
                               "}} - N_{\\text{",
                               country.id,"}}(t)\\right)\\cdot\\left(")
        for(j in 1:length(matrix.scrape)) {
          string.start <- paste0(string.start,
                                  " \\left(",
                                  matrix.scrape[j],
                                  "\\cdot 10^{-9}\\right)\\cdot N_{\\text{",
                                  names.scrape[j],
                                  "}}(t)+")
        }
        string.result <- substr(string.start, start = 1L, stop = nchar(string.start) - 1)
        string.result <- paste0(string.result, "\\right)$$")
        return(string.result)
      } else {
        return("No model equation available if parameters are either all too small or if the country do
      }
    },
    error = function(e) {
      return("No model equation available if parameters are either all too small or if the country does
    })
}
```

Shiny app code

This section displays the source code to create the Shiny dashboard displayed in the graphics in section 2.

```
### Dashboard

# header

header <- dashboardHeader(title = "CoViD-19")

# sidebar

sidebar <- dashboardSidebar(
  sidebarMenu(id = "tabs",
    menuItem("Intro", tabName = "IntroTab", icon = icon("microscope", lib = "font-awesome")),
    menuItem("Cases", tabName = "covidcases", icon = icon("ambulance", lib = "font-awesome")),
    menuItem("Deaths", tabName = "coviddeaths", icon = icon("skull", lib = "font-awesome"))
  )
)

# body

body <- dashboardBody(
  tabItems(
    tabItem(tabName = "IntroTab", h2(withMathJax(sprintf("Introduction to the CoViD-19 dashboard"))),
            fluidRow(
              box(
                title = "Dashboard Read-Me: Background, results, further comments",
                withMathJax(HTML(intro.string.1)),
                withMathJax(HTML(intro.string.2)),
                withMathJax(HTML(intro.string.3)),
                withMathJax(HTML(intro.string.4)),
                withMathJax(HTML(intro.string.5)),
                withMathJax(HTML(intro.string.6)),
                withMathJax(HTML(intro.string.7)),
                withMathJax(HTML(intro.string.8)),
                withMathJax(HTML(intro.string.9)),
                width = 12,
                collapsible = TRUE
              )
            ),
            fluidRow(
              box(
                title = "Adjacent countries in CoViD-19 data provided by the ECDC",
                plotOutput("plot", width = "790px", height = "790px"),
                width = 6,
                collapsible = TRUE
              ),
              box(
                title = "Cases-simulation: A selection of countries",
                DT::dataTableOutput("casesdict", height = "750px"),
                width = 3,
                height = "810px",
                collapsible = TRUE
              ),
              box(
                title = "Deaths-simulation: A selection of countries",
                DT::dataTableOutput("deathsdict", height = "750px"),
                width = 3,
                height = "810px",
                collapsible = TRUE
              )
            )
            ),
    tabItem(tabName = "covidcases", h3("CoViD-19 cases throughout the world"),
            fluidRow(
              box(
                title = "Model equation for selected country",
                withMathJax(),
                uiOutput("caseseq"), width = 12)
            ),
      fluidRow(
        box(
          title = "Choose a country",
          selectInput("countrychoice1",
                      "Please select a country from the list below:",
                      choices = get_display_id(),
                      selected = "DE - Germany"),
          width = 3,
          collapsible = TRUE
        ),
        box(
          title = "Choose a date:",
          dateInput("date1", label = "Input a date",
                    value = Sys.Date() - 1,
                    min = get_min_date(),
                    max = get_max_date()),
          width = 3,
          collapsible = TRUE
        ),
        box(
          title = "Choose either cumulative or daily mode:",
          radioButtons("radio1", label = "Choose plotting mode",
                       choices = c("Cumulative" = "CumCases", "Daily" = "Cases"),
                       inline = TRUE),
          width = 3,
          collapsible = TRUE
        ),
        box(
          title = "Choose a duration for the forecast:",
          selectInput("forecastchoice1",
                      "Please select a duration in days from the list below:",
                      choices = c(1:14),
                      selected = 5),
          width = 3,
          collapsible = TRUE
        )
      ),
      fluidRow(
        box(highchartOutput("hccovid1"), width = 6),
        box(highchartOutput("hmcovid1"), width = 6,
            tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js"))
      ),
      fluidRow(
        box(highchartOutput("hmcovid1f"), width = 6,
            tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js")),
        box(highchartOutput("hccovid1f"), width = 6)
      ),
      fluidRow(
        box(title = "Parameter matrix: Levelplot",
            plotOutput("casesparamsplot", width = "790px", height = "790px"), width = 6),
        box(title = "Parameter matrix: Numerical entries",
            DT::dataTableOutput("casesparamstable", width = "790px", height = "750px"),
            width = 6,
            height = "810px")
      )
      ),
    tabItem(tabName = "coviddeaths",
            h3("CoViD-19 deaths throughout the world"),
            fluidRow(
              box(
                title = "Model equation for selected country",
                withMathJax(),
                uiOutput("deathseq"), width = 12)
              ),
            fluidRow(
              box(
                title = "Choose a country",
                selectInput("countrychoice2",
                            "Please select a country from the list below:",
                            choices = get_display_id(),
                            selected = "DE - Germany"),
                width = 3,
                collapsible = TRUE
              ),
              box(
                title = "Choose a date:",
                dateInput("date2", label = "Input a date",
                          value = Sys.Date() - 1,
                          min = get_min_date(),
                          max = get_max_date()),
                width = 3,
                collapsible = TRUE
              ),
              box(
                title = "Choose either cumulative or daily mode:",
                radioButtons("radio2", label = "Choose plotting mode",
                             choices = c("Cumulative" = "CumDeaths", "Daily" = "Deaths"),
                             inline = TRUE),
                width = 3,
                collapsible = TRUE
              ),
              box(
                title = "Choose a duration for the forecast:",
                selectInput("forecastchoice2",
                            "Please select a duration in days from the list below:",
                            choices = c(1:14),
                            selected = 5),
                width = 3,
                collapsible = TRUE
              )
            ),
            fluidRow(
              box(highchartOutput("hccovid2"), width = 6),
              box(highchartOutput("hmcovid2"), width = 6,
                  tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js"))
            ),
            fluidRow(
              box(highchartOutput("hmcovid2f"), width = 6,
                  tags$script(src = "https://code.highcharts.com/mapdata/custom/world-lowres.js")),
              box(highchartOutput("hccovid2f"), width = 6)
            ),
            fluidRow(
              box(title = "Parameter matrix: Levelplot",
                  plotOutput("deathsparamsplot", width = "790px", height = "790px"), width = 6),
              box(title = "Parameter matrix: Numerical entries",
                  DT::dataTableOutput("deathsparamstable", width = "790px", height = "750px"),
                  width = 6,
                  height = "810px")
            )
      )
)
)
```

```
# UI

ui <- dashboardPage(
  header,
  sidebar,
  body,
  title = "CoViD-19 - Shiny Dashboard by David T. Heider",
  skin = "black"
)

# Server function

server <- function(input, output, session) {

  # Server logic for tab 1

  # Server logic for adjacency matrix plot on tab 1

  output$plot <- renderPlot({
    heatmap(adjacency.matrix, Rowv = NA, symm = TRUE, col= colorRampPalette(brewer.pal(8, "Blues"))(25)
  })

  # Server logic for cases dict data table on tab 1

  output$casesdict = DT::renderDataTable({
    DT::datatable(get_name_dictionary(selected.mode = "Cases"),
                  fillContainer = TRUE,
                  height = "750px",
                  options = list(lengthMenu = c(5, 10, 15, 30, 50),
                                 pageLength = 15))
  })

  # Server logic for deaths dict data table on tab 1

  output$deathsdict = DT::renderDataTable({
    DT::datatable(get_name_dictionary(selected.mode = "Deaths"),
                  fillContainer = TRUE,
                  height = "750px",
                  options = list(lengthMenu = c(5, 10, 15, 30, 50),
                                 pageLength = 15))
  })

  # Server logic for tab 2


  # Server logic for time-series plot on tab 2

  country.id1 <- reactive({
    substring(input$countrychoice1, 1, 2)
  })

  output$hccovid1 <- renderHighchart({
    plot_CoViD_cases(country.id = country.id1())
```

```r
})

# Server logic for equation on tab 2

output$caseseq <- renderUI({
  withMathJax(helpText(get_model_equation(selected.mode = "Cases", selected.country = country.id1()))
})
 # Server logic for chloropleth on tab 2

date.id1 <- reactive({
  input$date1
  })

selected.value1 <- reactive({
  input$radio1
})


output$hmcovid1 <- renderHighchart({
    geo_plot_CoViD_data(date.id = date.id1(), selected.value = selected.value1())
})

# Server logic for forecast time series plot and chloropleth on tab 3

duration1 <- reactive({
  input$forecastchoice1
})

output$hccovid1f <- renderHighchart({
  plot_simulated_CoViD_cases(duration = duration1(),
                              selected.country = substr(country.id1(), start = 1, stop = 2))
})

output$hmcovid1f <- renderHighchart({
  geo_plot_simulated_CoViD_data(duration = duration1(),
                                selected.mode = "Cases")
})

# Server logic for plot on tab 2

output$casesparamsplot <- renderPlot({
  plot_CoViD_param_matrix(selected.mode = "Cases")
})

# Server logic for data table on tab 2

output$casesparamstable = DT::renderDataTable({
  DT::datatable(produce_CoViD_param_matrix(selected.mode = "Cases"),
                fillContainer = TRUE,
                height = "750px",
                options = list(lengthMenu = c(5, 10, 11, 30, 50),
                               pageLength = 11))
})
```

```r
# Server logic for tab 3

# Server logic for time series plot on tab 3

country.id2 <- reactive({
  substring(input$countrychoice2, 1, 2)
})

output$hccovid2 <- renderHighchart({
  plot_CoViD_deaths(country.id = country.id2())
})

# Server logic for equation on tab 3

output$deathseq <- renderUI({
  withMathJax(print(get_model_equation(selected.mode = "Deaths", selected.country = country.id2())))
})
# Server logic for chloropleth on tab 3

date.id2 <- reactive({
  input$date2
})

selected.value2 <- reactive({
  input$radio2
})

output$hmcovid2 <- renderHighchart({
  geo_plot_CoViD_data(date.id = date.id2(), selected.value = selected.value2())
})

# Server logic for forecast time series plot and chloropleth on tab 3

duration2 <- reactive({
  input$forecastchoice2
})

output$hccovid2f <- renderHighchart({
  plot_simulated_CoViD_deaths(duration = duration2(),
                              selected.country = substr(country.id2(), start = 1, stop = 2))
})

output$hmcovid2f <- renderHighchart({
  geo_plot_simulated_CoViD_data(duration = duration2(),
                                selected.mode = "Deaths")
})

# Server logic for plot on tab 3

output$deathsparamsplot <- renderPlot({
  plot_CoViD_param_matrix(selected.mode = "Deaths")
})
```

```r
  # Server logic for data table on tab 3

  output$deathsparamstable = DT::renderDataTable({
    DT::datatable(produce_CoViD_param_matrix(selected.mode = "Deaths"),
                  fillContainer = TRUE,
                  height = "750px",
                  options = list(lengthMenu = c(5, 10, 11, 30, 50),
                                 pageLength = 11))
  })
}

### Build App

shinyApp(ui, server)
```