University of the Aegean
School of Engineering
Department of Financial and Management Engineering

Thesis Paper

Model and Analysis of the active reproduction rate(Rt) of COVID-19 cases by geographic region in Greece

Kostoulis Tilemachos
2312015055
Management Engineering
fme15055@fme.aegean.gr

Department of Financial & Management Engineering
in the School of Engineering
University of the Aegean January, 2023

Abstract

This thesis presents a study on the active reproduction rate (Rt) of COVID-19 cases in Greece by geographic region. A model developed by Kevin Systrom and his team, which uses the likelihood function and Bayes' theorem, is utilized to calculate the Rt values. This model uses the number of cases and the serial interval to estimate Rt. The study then builds a web application using Python programming language to visualize the data on the screen. The web application calculates the Rt values for each day and creates a 95% highest density interval. The visualization provides a clear representation of the Rt values for each region and helps in monitoring the spread of the virus. The results of this study provide a comprehensive understanding of the dynamics of the COVID-19 outbreak in Greece and will aid in the development of effective public health strategies.

Acknowledgements

I would like to thank Mr. Kevin Systrom and his team for their invaluable contributions to the understanding of the spread of COVID-19. Their model, which uses Bayes' Theorem and posteriors to calculate the rate of effective reproduction Rt of the virus, has been extremely valuable in helping us understand the dynamics of the outbreak and inform public health strategies to control its spread. Their dedication and hard work in developing this model and making it available to the public is greatly appreciated and has greatly contributed to our collective efforts to combat the pandemic.

Table of Contents

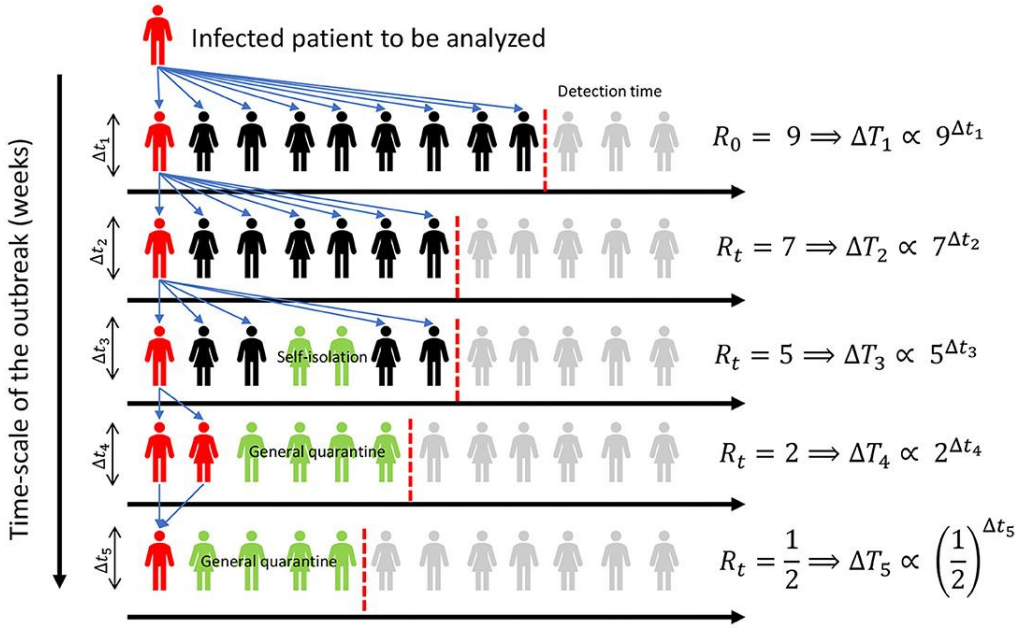
Introduction

In any pandemic, there are 3 important factors to consider: the number of people infected, the type and severity of infection, and the rate of reproduction of the virus.

The rate of effective reproduction, also known as the reproductive number or R_0, is a measure of the contagiousness of a disease. It is defined as the average number of people that an affected individual infects in a population in which everyone is susceptible to the disease. The effective reproductive rate is an important concept in epidemiology because it determines the potential spread of a disease and can help public health officials make decisions about how to control the spread of infection.

In this paper, we will analyze how the dynamic effective reproduction rate (Rt) is calculated by taking a closer look at the model created by Kevin Systrom's team while also producing visualization for each geographic region of Greece. The visualization is created using a custom web application in the Python programming language. Below is a step-by-step description of how this web application works.

COVID-19 caused by the novel coronavirus SARS-CoV-2, which is a highly contagious disease that has significant impact on global public health. The rate of effective reproduction for COVID-19 is not a fixed value and may vary depending on a number of factors, including the behavior of infected individuals, the effectiveness of public health interventions, and the level of immunity in the population.



(In Progress)

One widely used method for estimating the R0 for COVID-19 is the exponential growth model, which assumes that the number of cases grows exponentially in the early stages of an outbreak. This model can be used to estimate the R0 by fitting a curve to the early data on the growth of cases and using the slope of the curve to estimate the rate of growth.

Another method that has been used to estimate the R0 for COVID-19 is the likelihood approach, which uses statistical techniques to fit a model to the data on the spread of the disease. This method allows for the incorporation of more complex factors that can affect the R0, such as the timing of interventions and the effectiveness of infection control measures.

Overall, the R0 for COVID-19 is a critical measure for understanding the spread of the virus and for developing strategies to control its transmission. While the exact value of the R0 can vary depending on a range of factors, estimates from different methods suggest that it is typically between 2 and 3, indicating that the virus can spread quickly and easily in a susceptible population.

(In Progress) Difference Between R0 and Rt

R0 and Rt are both measures of the transmission rate of a disease. R0, also known as the basic reproduction number, is the average number of people that an infected person will infect in a population where everyone is susceptible to the disease. Rt, also known as the effective reproduction number, is the average number of people that an infected person will infect in a population where some individuals are immune to the disease or are otherwise protected from infection.

R0 is a measure of the potential spread of a disease in a population, while Rt is a measure of the actual spread of a disease in a population. R0 is typically estimated using mathematical models or by analyzing data from past outbreaks, while Rt is typically estimated using data on the current spread of a disease.

The main difference between R0 and Rt is that R0 is a theoretical measure of disease transmission, while Rt is a measure of disease transmission that takes into account the effects of immunity and other protective factors. R0 is a constant for a given disease, while Rt can vary over time and can be affected by interventions such as vaccination or social distancing.

In summary, R0 and Rt are both measures of the transmission rate of a disease, but R0 is a theoretical measure of potential spread, while Rt is a measure of actual spread that takes into account the effects of immunity and other protective factors.

Examples - Static Rt vs Dynamic Rt

Static Rt Spread (R0)

Assume a virus has a Rt of 2. This means that each infected person will infect two other people on average. Assume that the virus begins with just one infected person in a population of 100.

On Day 1, the infected person will have infected two more people, so there will be a total of three infected people.

On Day 2, each of the three infected people will have infected two more people, so there will be a total of nine infected people.

On Day 3, each of the nine infected people will have infected two more people, so there will be a total of 27 infected people.

And so on. As you can see, the number of infected people grows exponentially with each passing day, and the Rt value determines how quickly the virus spreads.

Rt=2
Day 1: 1 person infects 2 others, 3 infected total
Day 2: 3 people infect 2 others each for a total of 6 new infections, 9 infected total
Day 3: 9 people infect 2 others each for a total of 18 new infections, 27 infected total

Dynamic Rt Spread

Assume a virus has a Rt of 2. This means that each infected person will infect two other people on average. Assume that the virus begins with just one infected person in a population of 100.

On Day 1, the infected person will have infected two more people, so there will be a total of three infected people. However, let's say that the Rt value has now increased to 3, meaning that each infected person will now infect three other people on average.

On Day 2, each of the three infected people will have infected three more people, so there will be a total of 12 infected people.

On Day 3, each of the 12 infected people will have infected three more people, so there will be a total of 48 infected people.

And so on. As you can see, the Rt value can change over time, which can affect the rate at which the virus spreads. A higher Rt value means that the virus will spread more quickly, while a lower Rt value means that the virus will spread more slowly.

Day 1: Rt=2, 1 person infects 2 others, 3 infected total
Day 2: Rt=3, 3 people infect 3 others each for a total of 9 new infections, 12 infected total
Day 3: Rt=3, 12 people infect 3 others each for a total of 36 new infections, 48 infected total

Model

In-depth Analysis

Bayes Theorem

Every day, the number of new cases of COVID-19 provides us with valuable information about the current value of the reproduction number Rt. This metric helps us understand the current transmission rate of the disease and is calculated using Bayes' Theorem to find the distribution. By analyzing the new case count, we can update our understanding of Rt and how it has changed over time. We also know that the current value of Rt is related to the value from the previous day and all previous days, giving us a fuller picture of the spread of the disease. By tracking Rt over time, we can make more informed decisions about how to respond to the pandemic and protect public health.

What is Bayes' Theorem?

https://charanhu.medium.com/naive-bayes-algorithm-2a9415e21034

Bayes' Theorem, also known as Bayes' Rule, is a statistical formula that allows us to update our belief about an event based on new evidence. It is named after Thomas Bayes, a statistician who developed the theorem in the 18th century.

In particular, Bayes' Theorem can be employed to compute the likelihood of an event based on previous knowledge and new data. It is widely used in fields like statistics, machine learning and computer science to make predictions and decisions using data. This theorem is utilized here in the same way like so:

$$P(R_t|k) = \frac{P(k|R_t) \cdot P(R_t)}{P(k)} \qquad (1)$$

Having seen k new cases, it is believed that the distribution of Rt is equal to:

- The likelihood of seeing k new cases given Rt times $P(k|R_t)$
- The prior beliefs of the value of P(Rt) without the data $P(R_t)$
- Divided by the probability of seeing this many cases in general $P(k)$

Likelihood Function

The likelihood function is a function that describes the probability of observing a given set of data, given a specific set of parameters. It is typically expressed as a probability density or probability mass function (PMF), depending on the type of data and the type of hypothesis.. In other words, the likelihood function is a measure of how likely it is that a particular set of data (in this case k new cases) was generated by the model, with a given set of parameters, in this case λ or lambda and later Rt (see below for the connection of λ and Rt).

As it is believed that the Poisson Distribution is the best way to model arrivals, the Poisson Distribution's PMF is used here.

The form of the likelihood function is:

$$P(k|\lambda) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad (2)$$

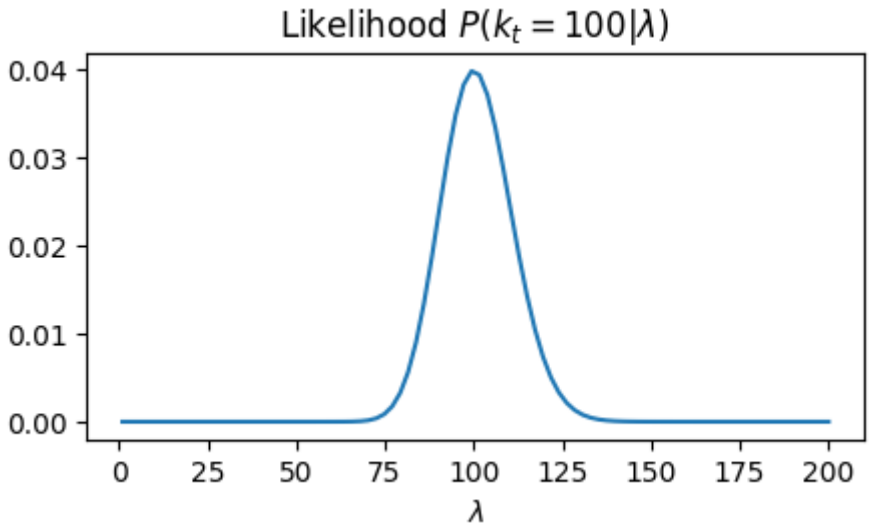
- the specific set of data k, for k new cases

- the parameter of the model λ

The Poisson distribution is a statistical model that describes the likelihood of a given number of events occurring within a given time interval, given a known average rate of occurrence. It can be used to calculate the likelihood of a specific number of events occurring based on the expected rate.

In the context of COVID-19, we can use the Poisson distribution to understand the likelihood of a certain number of new cases occurring based on the expected transmission rate (represented by the rate parameter, lambda). If we fix the number of observed cases (k) and vary the value of lambda, we can calculate the likelihood function, which represents the probability of each value of lambda given the observed number of cases.

For example, if we observe 100 new cases (Plot 1) and want to know how likely different lambda values are, we can use the Poisson distribution to calculate the probability of seeing 100 cases for each lambda value. The most likely value of lambda is 100, but other values could have produced 100 cases by chance. The likelihood function, given the observed data, allows us to understand the probability of each value of lambda.



Plot 1: Likelihood for fixed k=100 cases

(Make it fuller) Connection of λ and Rt

From the Bettencourt & Ribeiro article:
https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0002185
To find the expression accounting for the evolution of new cases $\Delta T(t + \tau)$ the Equation:

$$\frac{dS}{dt} = -\beta\frac{S}{N}I \quad \frac{dI}{dt} = \beta\frac{S}{N}I - \gamma I$$

[standard epidemic susceptible-infected reproduction (SIR) model]

has been integrated for I(t) between t and $t + \tau$, to obtain

$$I(t + \tau) = I(t)exp[\gamma \int_{t}^{t+\tau} (R_0 \frac{S(t')}{N(t')} - 1)] = I(t)exp[\tau\gamma(R_t - 1)] \quad \text{for} \quad R_t = R_0 \frac{S(t')}{N(t')}$$

So, it stands to reason for this to make sense for I(t) between $t - 1$ and t:

$$I(t) = I(t - 1)exp[\gamma(R_t - 1)$$

By plugging in the λ and k values we get:

$$\lambda(t) = k(t - 1)exp[\gamma(R_t - 1) \quad \Rightarrow \quad \lambda_t = k_{t-1}e^{\gamma(R_t - 1)}$$

Reciprocal of the Serial Interval, Gamma(γ)

What is the reciprocal of the serial interval, gamma(γ)?

The serial interval, or the time between the appearance of symptoms in a primary case and the appearance of symptoms in a secondary case infected by the primary case, is an important metric for understanding the transmission dynamics of a disease. The reciprocal of the serial

interval, or its inverse, is defined as $\frac{1}{T}$, where T is the serial interval. The reciprocal of the serial interval, which is a measure of the time between successive cases in a chain of transmission, is estimated to be in the range of 0.11 to 0.14 (1/9 to 1/7). This estimate is used in statistical models to determine the rate at which the virus spreads from person to person and to inform public health strategies for controlling its spread. By determining the reciprocal of the serial interval, it is possible to understand the rate of transmission of the virus.

In other words, the reciprocal of the serial interval is an important quantity in the study of the spread of infectious diseases, and it is related to the growth rate of a disease.

$\gamma = \frac{1}{7}$ based on the Estimating R_0 paragraph from the article below:

https://wwwnc.cdc.gov/eid/article/26/7/20-0282_article

More specifically the serial interval is estimated to be 7-8 days based on data from Wuhan and Shenzhen, China therefore the reciprocal is 1/7.

Applying Bayes' Theorem

(In Progress) Posteriors

The cumprod method is often used in statistical analysis and machine learning to calculate the cumulative product of a series of values. In this case, it may be used to calculate the cumulative product of the likelihood of different values of the reproduction number (Rt) over time, which can be used to update our belief about the current value of Rt based on new evidence.

The posterior distribution plays a crucial role in Bayesian statistics, providing a means of incorporating prior knowledge and evidence to update our beliefs about the likelihood of different hypotheses. By taking into account both the prior probabilities and the likelihood of the evidence given each hypothesis, the posterior distribution allows us to make more informed decisions and predictions. In this thesis, we will explore the definition, properties, and applications of the posterior distribution, and discuss how it can be used to make more accurate and reliable inferences.

The posterior distribution is a probability distribution that represents our belief about the likelihood of different hypotheses or parameters after taking into account the prior probabilities and the evidence.

It is calculated using Bayes' theorem, which states that the posterior probability of a hypothesis A given evidence B is equal to the prior probability of the hypothesis multiplied by the likelihood of the evidence given the hypothesis, divided by the probability of the evidence.

$$P(R_t|k) = \frac{P(k-1|R_t) \cdot P(R_t)}{P(k)}$$

Highest Density Interval

A Highest Density Interval (HDI) is a statistical concept that describes a range of values within a dataset that contains the highest concentration of observations. HDIs are often used in Bayesian statistics to represent a more detailed understanding of the uncertainty associated with a model or estimate by providing a range of values rather than a single point estimate.

One way to calculate an HDI is to first compute the probability density function (PDF) for the dataset in question. The PDF describes the likelihood of observing a particular value within the dataset, and can be thought of as a smooth curve that describes the distribution of the data.

Once the PDF has been calculated, the next step is to find the interval within the dataset that contains the highest concentration of observations. This is typically done by finding the area under the PDF curve that contains a specified percentage of the observations (e.g. 95%).

Once the interval has been identified, the lower and upper bounds of the interval can be used to represent the uncertainty associated with the estimate. For example, if an HDI was calculated for a dataset with a mean of 10 and a standard deviation of 2, the resulting HDI might be 8 to 12, indicating that there is a 95% probability that the true population parameter falls within that range.

(In Progress) Choosing the Gaussian standard deviation σ for $P(R_t|R_{t-1})$

The original method for choosing the prior distribution for our model is to use the previous day's posterior as the current prior. This method, however, does not account for the possibility that the value of Rt has changed since the previous day. To address this issue, Gaussian noise with a specified standard deviation is applied to the prior distribution. The higher the standard deviation, the more noise is used and the more the value of Rt is expected to change every day. This iterative noise application has the effect of naturally decaying distant posteriors, similar to windowing, but it is more robust and does not discard posteriors arbitrarily after a certain period of time. The new approach computes a series of values that explain all of the cases, assuming that the value of Rt varies by a certain amount each day, whereas the previous windowing approach computed a fixed value at each time point that explained the cases' surrounding days.

What is windowing?

https://wiki.aalto.fi/display/ITSP/Windowing

Windowing is a technique used in signal processing, time series analysis and other fields to split a signal into overlapping or non-overlapping segments. Each segment is called a

window, and the goal is to extract features, patterns or other information from each window. The window functions can be applied on the data to reduce noise, or to emphasize certain features of the signal. Depending on the application, different types of window functions can be used, such as rectangular, triangular, Hann, Hamming, Blackman, etc.

In the context of Bayesian analysis, windowing may refer to a method of discarding or downweighting past observations or data points in order to give more weight to more recent information. This is used as an alternative to other methods, such as applying noise to the prior distribution to account for changes in the value of Rt over time.

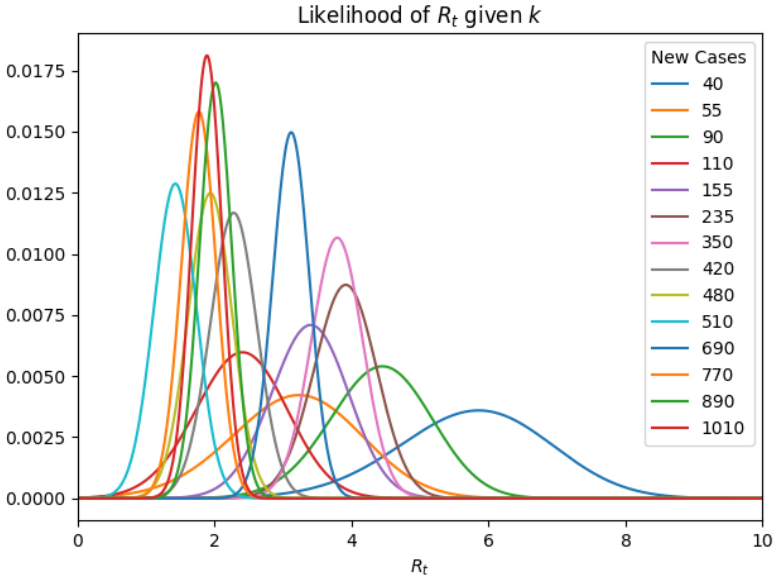
(In progress)
Adam Lerer pointed out that we can use the process of maximum likelihood to inform our choice

Example Rt of a 2 week period

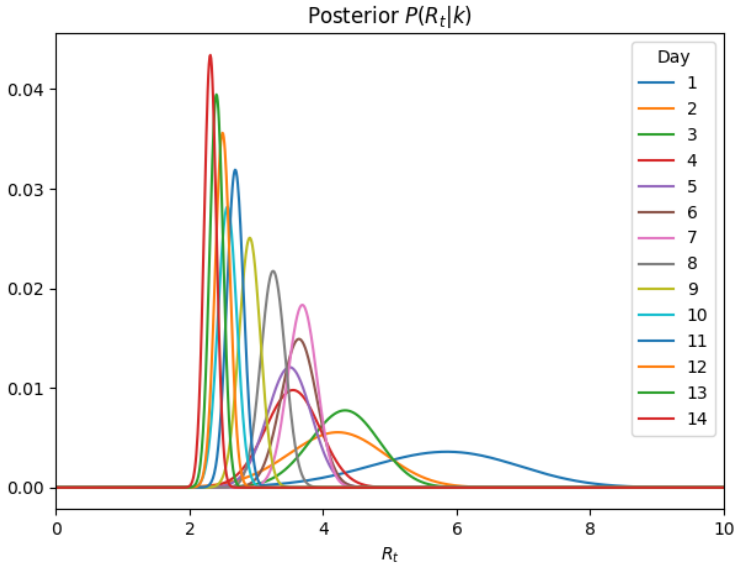
Let's say we want to make a random example for a 2 week period of a city. For 14 days we have received the following data

Day	1	2	3	4	6	6	7	8	9	10	11	12	13	14
Cases	40	15	35	20	45	80	115	70	60	30	180	80	120	120

If we calculate the likelihood of Rt for any given day, the likelihood distribution for each individual day would look like this



If we take into account the posterior of the previous day the new distribution for each day would look like this



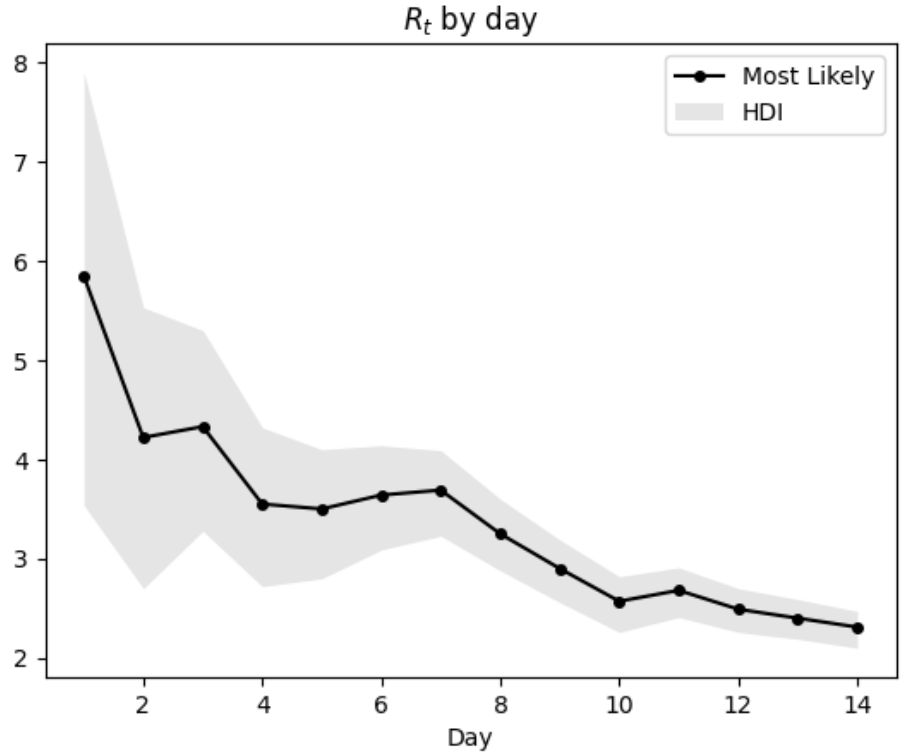
The following is the Most Likely Rt for each day

Day	1	2	3	4	6	6	7	8	9	10	11	12	13	14
Rt	5.85	4.22	4.33	3.55	3.50	3.64	3.69	3.25	2.90	2.57	2.68	2.49	2.40	2.31

After calculating a 95% HDI with a Lower Bound and a Higher Bound

Day	1	2	3	4	6	6	7	8	9	10	11	12	13	14
Rt	5.85	4.22	4.33	3.55	3.50	3.64	3.69	3.25	2.90	2.57	2.68	2.49	2.40	2.31
L95	3.54	2.70	3.28	2.72	2.80	3.09	3.23	2.88	2.56	2.26	2.41	2.26	2.19	2.10
H95	7.90	5.53	5.30	4.32	4.10	4.14	4.09	3.60	3.19	2.82	2.91	2.70	2.59	2.47

The information above would look like this



(In Progress) Code

Running the algorithm with real data

Initialization

```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
from matplotlib.dates import date2num, num2date
from matplotlib import dates as mdates
from matplotlib import ticker
from matplotlib.colors import ListedColormap
from matplotlib.patches import Patch

from scipy import stats as sps
from scipy.interpolate import interp1d

from IPython.display import clear_output

import datetime as dt
```

This code imports several libraries that are commonly used in data analysis and visualization.

- **pandas** is a library used for manipulating and analyzing data, specifically in the form of a DataFrame.
- **numpy** is a library for working with arrays and performing numerical operations.
- **matplotlib** is a library for creating visualizations in Python, including static, animated, and interactive plots. It has a **pyplot** module that mimics the plotting functions of MATLAB. There are also submodules within matplotlib such as date2num, num2date, mdates, ticker and ListedColormap that are used for various plotting needs, such as converting between date and number or formatting date and time.

- **scipy** is a library for scientific and technical computing that includes a stats module for statistical operations, as well as the interp1d function for interpolation.
- **IPython.display** is a module of IPython that allows for displaying rich output in Jupyter notebooks, and clear_output is a function for clearing the output of a notebook cell.
- The **datetime** module in the Python standard library provides classes for working with dates and times.

(In progress) Get data

[Insert code for fetching data]
[Insert explanation of code]

Prepare cases, turn into a smoothed curve

```
state_name = 'Greece'

def prepare_cases(cases, cutoff=25):
    new_cases = cases.diff()

    smoothed = new_cases.rolling(7,
        win_type='gaussian',
        min_periods=1,
        center=True).mean(std=2).round()

    idx_start = np.searchsorted(smoothed, cutoff)

    smoothed = smoothed.iloc[idx_start:]
    original = new_cases.loc[smoothed.index]

    return original, smoothed

cases = states.xs(state_name).rename(f"{state_name} cases")

original, smoothed = prepare_cases(cases)
```

This code defines a function called "prepare_cases" that takes in two arguments: "cases" and "cutoff". The "cases" argument is used to calculate the new cases by taking the difference of the original cases. Then it creates a smoothed version of the new cases using the "rolling" method, which applies a "gaussian" window to smooth out the data. The window size is 7, and the minimum number of periods required for the window is 1. The "center" parameter is set to True, which means that the window will be centered on each data point. The "std" parameter is set to 2, which controls the standard deviation of the gaussian window.

The function then searches for the first index where the smoothed cases are greater than the cutoff value. The resulting index is used to slice the smoothed cases and the original new cases.

The state_name variable is set to 'Greece' as a string.
It then uses the xs method to select the cases of the state_name from the states DataFrame.
Then it renames the cases with the state name and original and smoothed cases are returned.

Steps for calculating the posteriors and Most Likely Rt values

```
# We create an array for every possible value of Rt
R_T_MAX = 12
r_t_range = np.linspace(0, R_T_MAX, R_T_MAX*100+1)

GAMMA=1/7

def get_posteriors(sr, sigma=0.15):

    # (1) Calculate Lambda
    lam = sr[:-1].values * np.exp(GAMMA * (r_t_range[:, None] - 1))

    # (2) Calculate each day's likelihood
    likelihoods = pd.DataFrame(
        data = sps.poisson.pmf(sr[1:].values, lam),
        index = r_t_range,
        columns = sr.index[1:])
```

```
# (3) Create the Gaussian Matrix
process_matrix = sps.norm(loc=r_t_range,
                          scale=sigma
                         ).pdf(r_t_range[:, None])

# (3a) Normalize all rows to sum to 1
process_matrix /= process_matrix.sum(axis=0)

# (4) Calculate the initial prior
#prior0 = sps.gamma(a=4).pdf(r_t_range)
prior0 = np.ones_like(r_t_range)/len(r_t_range)
prior0 /= prior0.sum()

# Create a DataFrame that will hold our posteriors for each day
# Insert our prior as the first posterior.
posteriors = pd.DataFrame(
    index=r_t_range,
    columns=sr.index,
    data={sr.index[0]: prior0}
)

# We said we'd keep track of the sum of the log of the probability
# of the data for maximum likelihood calculation.
log_likelihood = 0.0

# (5) Iteratively apply Bayes' rule
for previous_day, current_day in zip(sr.index[:-1], sr.index[1:]):

    #(5a) Calculate the new prior
    current_prior = process_matrix @ posteriors[previous_day]

    #(5b) Calculate the numerator of Bayes' Rule: P(k|R_t)P(R_t)
    numerator = likelihoods[current_day] * current_prior

    #(5c) Calcluate the denominator of Bayes' Rule P(k)
    denominator = np.sum(numerator)

    # Execute full Bayes' Rule
    posteriors[current_day] = numerator/denominator

    # Add to the running sum of log likelihoods
    log_likelihood += np.log(denominator)
```

```
    return posteriors, log_likelihood

# Note that we're fixing sigma to a value just for the example
posteriors, log_likelihood = get_posteriors(smoothed, sigma=.25)
```

This code defines a function called "get_posteriors" that takes in two arguments "sr" and "sigma". The "sr" argument is a time-series data of the reported cases and "sigma" is the standard deviation used in the Gaussian distribution.

The function starts by initializing a variable R_T_MAX and creating an array "r_t_range" which will hold every possible value of Rt.
It also initializes a variable GAMMA which is set to 1/7.

Then, the function proceeds with several steps:

1. Calculates the value of Lambda by multiplying the values of the reported cases from the previous day with the exponential of GAMMA multiplied by the difference between Rt and 1
2. Calculates the likelihood of each day's case count using a Poisson distribution with lambda calculated in step 1
3. Creates a Gaussian matrix using the "loc" parameter set to "r_t_range" and "scale" parameter set to "sigma"
4. Normalizes all rows to sum up to 1
5. Calculate the initial prior by dividing the range of Rt by the length of Rt
6. Create a DataFrame that will hold the posteriors for each day and insert the prior as the first posterior
7. Iteratively apply Bayes' rule to calculate the new prior, numerator and denominator of Bayes' rule and execute full Bayes' rule
8. Add the running sum of log likelihoods

Finally, the function returns the posteriors and log_likelihood.
At the end of the code snippet, the function is called with the smoothed cases and sigma value of 0.25.

HDI Function

```
def highest_density_interval(pmf, p=.95, debug=False):
    # If we pass a DataFrame, just call this recursively on the columns
    if(isinstance(pmf, pd.DataFrame)):
            return pd.DataFrame([highest_density_interval(pmf[col], p=p)
for col in pmf],
                                index=pmf.columns)

    cumsum = np.cumsum(pmf.values)

    # N x N matrix of total probability mass for each low, high
    total_p = cumsum - cumsum[:, None]

    # Return all indices with total_p > p
    lows, highs = (total_p > p).nonzero()

    # Find the smallest range (highest density)
    best = (highs - lows).argmin()

    low = pmf.index[lows[best]]
    high = pmf.index[highs[best]]

    return pd.Series([low, high],
                     index=[f'Low_{p*100:.0f}',
                            f'High_{p*100:.0f}'])

hdi = highest_density_interval(posteriors, debug=True)
```

This code defines a function called "highest_density_interval" that takes in two arguments "pmf" and "p" where "pmf" represents the probability mass function of the data and "p" is a value between 0 and 1 that represents the probability of the interval. The "debug" argument is set to False by default, and if set to True, it will print some debug information.

The function first checks if "pmf" is a DataFrame, and if so, it calls the function recursively on the columns and returns a DataFrame with the same index as "pmf"

Then it calculates the cumulative sum of the pmf values and creates a matrix "total_p" that holds the total probability mass for each low and high value.
It then find the indices where the total probability is greater than p, and finds the smallest range (highest density) by subtracting the lows from the highs and getting the index of the minimum value.
It then assigns the low and high values using the indices, and returns a pandas series that contains the low and high values and index as "Low_p" and "High_p" where p is the probability passed to the function.

At the end of the code snippet, the function is called on the posteriors with p=0.95 and debug is set to False.
The returned value is assigned to the variable hdi.

Connecting ML with HDI

```
# Note that this takes a while to execute - it's not the most efficient
algorithm
hdis = highest_density_interval(posteriors, p=.95)

most_likely = posteriors.idxmax().rename('ML_Rt')

result = pd.concat([most_likely, hdis], axis=1)
```

This code calls the highest_density_interval function on the posteriors with p = 0.95 and assigns the returned value to the variable "hdis".
Then it finds the most likely value of Rt by finding the index of the maximum value in the posteriors and renaming it as "ML_Rt"
Then it concatenates the "most_likely" and "hdis" dataframe along the columns (axis=1) and assigns the resulting dataframe to the variable "result".

It is noted that the highest_density_interval function takes a while to execute as it's not the most efficient algorithm. It is likely that the function is iterating over a large amount of data in order to find the highest density interval which can be computationally expensive.

The shift -1 is probably used to align the index with the data.

Plotting

```
from calendar import MONDAY, TUESDAY


def plot_rt(result, ax, state_name):

    ax.set_title(f"{state_name}")

    # Colors
    ABOVE = [1,0,0]
    MIDDLE = [1,1,1]
    BELOW = [0,0,0]
    cmap = ListedColormap(np.r_[
        np.linspace(BELOW,MIDDLE,25),
        np.linspace(MIDDLE,ABOVE,25)
    ])
    color_mapped = lambda y: np.clip(y, .5, 1.5)-.5

    index = result['ML_Rt'].index.get_level_values('date')
    values = result['ML_Rt'].values

    # Plot dots and line
    ax.plot(index, values, c='k', zorder=1, alpha=.25)
    ax.scatter(index,
               values,
               s=40,
               lw=.5,
               c=cmap(color_mapped(values)),
               edgecolors='k', zorder=2)

    # Aesthetically, extrapolate credible interval by 1 day either side
    lowfn = interp1d(date2num(index),
                     result['Low_95'].values,
                     bounds_error=False,
                     fill_value='extrapolate')
```

```
    highfn = interp1d(date2num(index),
                      result['High_95'].values,
                      bounds_error=False,
                      fill_value='extrapolate')

    extended = pd.date_range(start=pd.Timestamp('2020-03-01'),
                             end=index[-1]+pd.Timedelta(days=1))

    ax.fill_between(extended,
                    lowfn(date2num(extended)),
                    highfn(date2num(extended)),
                    color='k',
                    alpha=.1,
                    lw=0,
                    zorder=3)

    ax.axhline(1.0, c='k', lw=1, label='$R_t=1.0$', alpha=.25);

    # Formatting
    ax.xaxis.set_major_locator(mdates.MonthLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%b'))
    ax.xaxis.set_minor_locator(mdates.DayLocator())

    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_formatter(ticker.StrMethodFormatter("{x:.1f}"))
    ax.yaxis.tick_right()
    ax.spines['left'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.margins(0)
    ax.grid(which='major', axis='y', c='k', alpha=.1, zorder=-2)
    ax.margins(0)
    ax.set_ylim(0.0, 5.0)
                                  ax.set_xlim(pd.Timestamp('2021-10-01'),
result.index.get_level_values('date')[-1]+pd.Timedelta(days=1))
    fig.set_facecolor('w')


fig, ax = plt.subplots(figsize=(600/72,400/72))

plot_rt(result, ax, state_name)
ax.set_title(f'Real-time $R_t$ for {state_name}')
```

```
ax.xaxis.set_major_locator(mdates.WeekdayLocator(byweekday=MONDAY,inter
val=1))
#
ax.xaxis.set_major_locator(mdates.DayLocator(bymonthday=range(1,32),int
erval=1)) # This is to show up every date per day on the plot
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %d %Y'))


plt.xticks(rotation = 90)

now = dt.datetime.now()
then = now + dt.timedelta(days=100)
days = mdates.drange(now,then,dt.timedelta(days=1))
```

(From test) Evaluating the Likelihood Function

```
k = np.array([20, 40, 55, 90])

# We create an array for every possible value of Rt
R_T_MAX = 12
r_t_range = np.linspace(0, R_T_MAX, R_T_MAX*100+1)

# Gamma is 1/serial interval
GAMMA = 1/7

# Map Rt into lambda so we can substitute it into the equation below
# Note that we have N-1 lambdas because on the first day of an outbreak
# you do not know what to expect.
```

```
lam = k[:-1] * np.exp(GAMMA * (r_t_range[:, None] - 1))

# Evaluate the likelihood on each day and normalize sum of each day to
1.0
likelihood_r_t = sps.poisson.pmf(k[1:], lam)
likelihood_r_t /= np.sum(likelihood_r_t, axis=0)

# Plot it
ax = pd.DataFrame(
    data = likelihood_r_t,
    index = r_t_range
).plot(
    title='Likelihood of $R_t$ given $k$',
    xlim=(0,10),
    figsize=(6,2.5)
)

ax.legend(labels=k[1:], title='New Cases')
ax.set_xlabel('$R_t$');
```

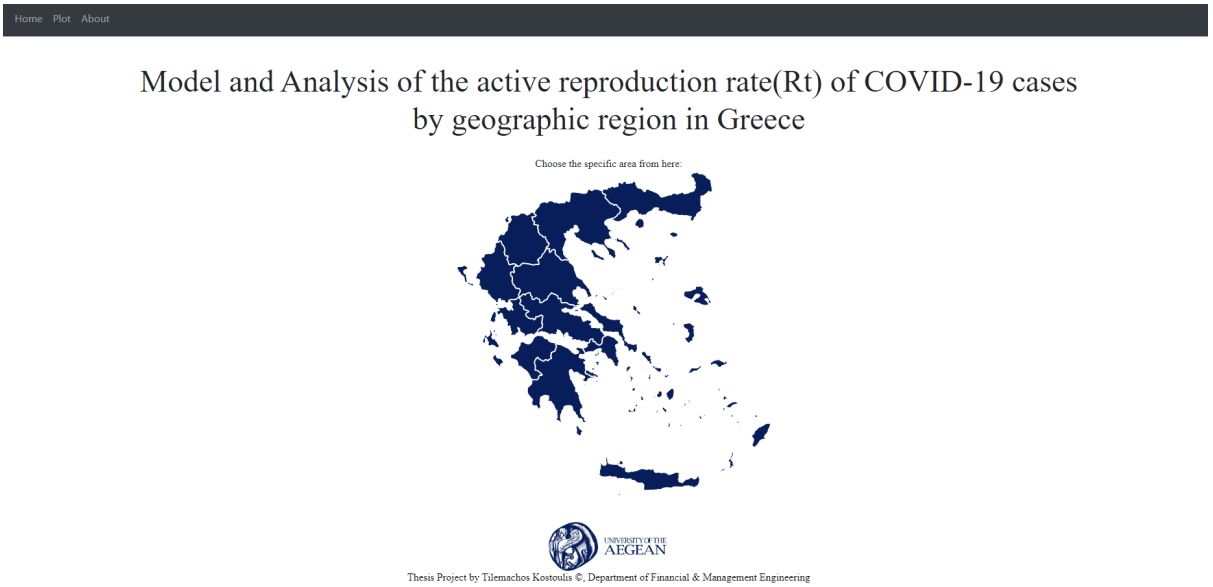
The code defines some constants and variables that will be used later in the script. k is an array of integers representing the number of new cases on different days. R_T_MAX is the maximum value that the variable Rt (which stands for the effective reproduction number) can take. r_t_range is an array of all possible values of Rt, ranging from 0 to R_T_MAX with a step size of 0.01. GAMMA is the reciprocal of the serial interval, which is the average time between when a person becomes infected and when they pass the infection on to someone else. lam is an array of values calculated from k and Rt using an exponential function.

The code then calculates the likelihood of each value of Rt given the observed number of new cases k, using a Poisson distribution. The likelihood is normalized so that the sum of the likelihoods for each day is equal to 1.0. Finally, the code uses the pandas library to plot the likelihoods using a matplotlib plot. The plot shows the likelihood of different values of Rt given the observed number of new cases. The legend shows the different values of k, and the x-axis shows the possible values of Rt.

Web Application

[Insert intro for web application]

Home



Plot

[Insert info and what is displayed on page]

About

[Insert info and what is displayed on page]

XAMPP / MariaDB

[Insert info]

SQL Map

[Insert info]

Helpful Info

- Probability mass functions (PMF) are used to describe discrete probability distributions. While probability density functions (PDF) are used to describe continuous probability distributions.

Appendix

- HDI = Highest Density Function
- PMF = Probability Mass Function
- PDF = Probability Density Function
- CDF = Cumulative Density Function
- Rt = Rate of Effective Reproduction at a specific time
- ML = Most Likely

References

Bayes Theorem: A classic reference for Bayes' Theorem is "An Introduction to Probability Theory and Its Applications" by William Feller. Another popular textbook is "Pattern Recognition and Machine Learning" by Christopher Bishop.

Likelihood function: A good reference for likelihood function is "Statistical Inference" by George Casella and Roger L. Berger, and "The Statistical Theory of Shape" by A. R. Brazzale and A. C. Davison.

Posterior distribution: A popular reference for understanding posterior distributions is "Bayesian Data Analysis" by Andrew Gelman, John Carlin, Hal Stern, David Dunson, Aki Vehtari, and Donald Rubin.

Highest density interval: A good reference for understanding the highest density interval is "Bayesian Data Analysis" by Andrew Gelman, John Carlin, Hal Stern, David Dunson, Aki

Vehtari, and Donald Rubin. Another good reference is "Probabilistic Forecasting and Bayesian Model Verification" by E.J. Kostelich and I. R. Epstein.

- Bayes' Theorem:
    - "An Introduction to Bayesian Analysis: Theory and Methods" by Jaynes, E.T
    - "The Theory of Probability" by Cox, R.T
    - "Bayesian Data Analysis" by Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.
- Likelihood function:
    - "Likelihood" by A.W.F. Edwards
    - "Likelihood Methods in Statistics" by A.C. Davison
    - "Statistical Inference" by George Casella and Roger L. Berger
- Posterior distribution:
    - "Markov Chain Monte Carlo in Practice" by W.R. Gilks, S. Richardson, D.J. Spiegelhalter
    - "Monte Carlo Statistical Methods" by Christian P. Robert and George Casella
    - "Practical Markov Chain Monte Carlo" by Steve Brooks, Andrew Gelman, Galin L. Jones, and Xiao-Li Meng
- Highest Density Interval:
    - "Efficient Computation of Highest Density Intervals" by T.J. Loredo and J.D. Phillips
    - "Bayesian Inference for the Statistical Analysis of Distributions" by J.M. Bernardo and A.F.M. Smith
    - "The Bayesian Bootstrap" by Bradley Efron

- Flask Web Development
    - Flask Web Development with Python Tutorial (pdf) by Antonio Mele
    - "Flask by Example" by Gareth Dwyer (book)
    - "Building Web Applications with Flask" by Gareth Dwyer (book)
    - "Full Stack Python Flask Guide" by Full Stack Python (Guide)
    - "Mastering Flask Web Development" by Jack Stouffer (book)