Training for infectious disease modelling with WHO Western Pacific Regional Offices

Table of contents

Overview

This document describes the development and application of an open-source software package called "summer" for infectious disease modelling. The software is designed to be user-friendly and accessible for epidemiologists and public health specialists, allowing them to build and calibrate compartmental models without needing extensive programming knowledge. The document highlights the software's key features, including its support for stratification, calibration using optimization algorithms and Bayesian sampling, and its ability to handle heterogeneous mixing and multiple strains.

Key Points:

- Summer is an open-source Python software package for compartmental infectious disease modelling designed to be user-friendly and accessible to epidemiologists and public health specialists.
- Summer simplifies the construction of compartmental models by providing a domain-specific syntax that maps clearly to the underlying epidemiological process.
- Estival, a complementary package, provides tools for calibrating and optimizing summer models, supporting both parameter optimization and Bayesian sampling.
- Summer supports the modelling of directly transmitted infectious diseases, including respiratory, gastrointestinal, and sexually transmitted infections.
- Summer models can be stratified by various factors, such as age, vaccination status, or location, to capture population heterogeneity.
- The software accounts for heterogeneous mixing patterns, where individuals do not interact randomly with each other.

- Summer can represent the progression of infection through multiple stages before an individual becomes infectious using serial latent compartments.
- Summer is highly customizable and can be applied to a range of different countries and local contexts.
- The software is designed to be easily integrated with other widely used tools in the Python ecosystem.

Background

The Epidemiological Modelling Unit (EMU), Monash University, Australia is a team of mathematicians, software developers, data scientists, epidemiologists, and public health specialists. We specialise in the development of infectious disease models that are used to inform decisions around public health policy and control. Throughout the COVID-19 pandemic, we have aided countries in the Asia-Pacific Region, particularly Malaysia and the Philippines through the support of the World Health Organization (WHO). We have also applied our expertise to support tuberculosis control across a number of country settings.To support this body of work we have developed our own open-source software in the programming language Python designed specifically for robust, flexible, and reproducible compartmental modelling of infectious diseases. Compartmental models are one of the most common approaches to infectious disease modelling. These models simulate the transmission and spread of infectious diseases at the population level. Published examples of the types of models we have built and applied to COVID-19 using our software can be found here for Malaysia, Philippines and Australia. This paper provides an example of our team applying these tools to tuberculosis in the Pacific Islands nation of the Marshall Islands.

Compartmental models in general are a tool that are a part of a suite of tools that can support high quality analysis of infectious disease data, delivering epidemiological insights. Our software is designed with the principle of making this tool more readily accessible for applied epidemiology and public health response.

Our core tool is called summer which allows the user to construct compartmental models that simulate infectious disease transmission. To support summer we have developed several other packages most notably estival, which is a package that supports the complex mathematics of calibration and optimization of compartmental models built in summer.

Why infectious disease modelling?

Mathematical modelling of Infectious diseases spans multiple techniques that can support surveillance and public health response to infectious diseases.

This modelling can take many forms, and there are now a vast array of computational techniques that can be applied to understanding infectious disease epidemiology and control. However, throughout this concept note, when we refer to infectious disease modelling, we are

explicitly referring to one of the most common types of infectious disease modelling known as compartmental modelling. Compartmental modelling is a form of simulation or mechanistic modelling. It is also a form of complex systems analysis, meaning that unanticipated behaviours of the system can emerge in the outputs of the simulation model. This type of model has an extensive history of research to support its implementation and is one of the most popular techniques in infectious disease modelling.

These models attempt to capture and represent the transmission process computationally and simulate how infectious diseases spread through the population. These models can be calibrated to local disease data, such as surveillance data generated on cases, hospitalisations and infection deaths, which helps to ensure the dynamics we observe in these settings have been accurately captured. Because they explicitly represent the key processes underlying transmission, models that are calibrated to local data can then be used to project the future trajectory of an epidemic, evaluate the timing in the peak of an epidemic and provide estimates of healthcare capacity utilisation. Furthermore, they can also estimate the impact of interventions and policies at the population level, particularly where they impact on transmission (e.g. mobility restrictions, vaccination programs or clinical treatment strategies).

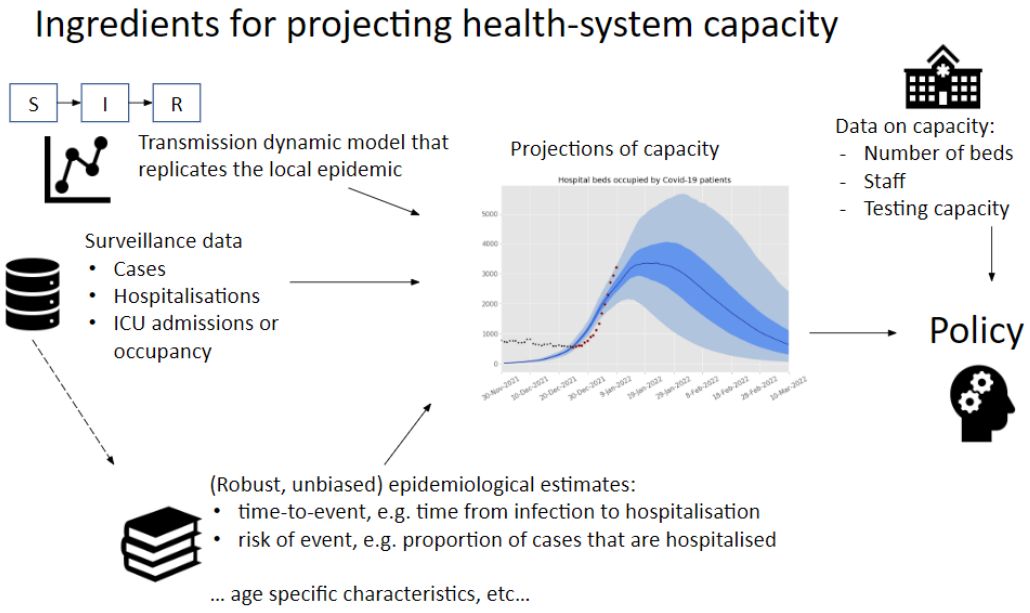


Figure 1: image.png

Figure 1: illustrates the integration of a dynamic transmission model with real-time surveillance data, robust epidemiological estimates, and health system capacity data. These elements

together help project future healthcare needs, guiding policy decisions to effectively manage the epidemic and mitigate its impact on the healthcare system.

Overview of *summer* and *estival*

Summer provides a domain-specific syntax that simplifies the construction of compartmental models of infectious disease transmission. That is the construction of models is developed in a way that maps clearly to the underlying epidemiological process that is occurring, this opposed to the alternative approach of expressing compartmental models in terms of the underlying mathematical structure - ordinary differential equations (ODEs). ***Summer*** is still based on ODEs but the computation and solving of the system of equations that gives us the complex outputs of a compartmental model is handled in the backend. This design feature is intended to lower the barrier to entry and reduce the learning curve for users. Users need to only understand the principles in order to design and run compartmental models and do not need detailed knowledge of applying complex ODE solvers using Python. An additional, flow-on feature is that it makes the code used to construct models easier to understand for those who may be less familiar with the technical aspects. This supports collaborative model development with epidemiologists and public health specialists who can provide important context and subject knowledge around the model disease, and play an important role in understanding and interpreting model results.

For information about summer, we have some resources below at the following links. 1. Documentation for summer: summer2 docs 2. Our Github repository: summer2 github 3. Our infectious disease modelling textbook: Textbook

Estival is a package that provides the tools for the calibration (and optimization) of ***summer*** compartmental models. In particular, it allows for the robust calibration of compartmental models that represent transmission of an infectious disease to the surveillance or epidemiological data for that disease, such as case notifications, seroprevalence surveys or hospitalizations. The methods employed for calibration typically focus on Bayesian inference methods, but just as ***summer*** handles the computation for solving the system of equations that govern the dynamics of the compartmental model, ***estival*** supports the integration of the models with the computational tools of a number other packages developed within the Python ecosystem. In essence ***estival*** provides a simple and easy to use interface to apply these methods to a ***summer*** model.

A simple comparison that can be drawn for ***summer*** (and ***estival***) is to consider it to be similar software packages that have been developed for handling common statistical models such as linear regression. Many people working in epidemiology and public health will be familiar with these software packages that handle the complex computational process of model fitting in the backend, allowing the user to focus on model specification and interpretation of results. Examples include the package R stats for programming language R and Scikit-learn in Python.

Both ***summer*** and ***estival*** are available free for download, as open-source continuously maintained and updated software libraries. They are both listed on the Python package index (PyPI), which helps to facilitate access to these packages by users. Summer is also supported by a dedicated webpage for documentation that details how to implement summer models and provides example code.

Google Colab

Google Colab is an online platform that allows us to run Python code in a collaborative environment. It provides a convenient way to work with Jupyter notebooks without having to install any software locally. During our session, we'll be using Google Colab extensively to interact with the notebooks and explore infectious disease modelling using Python and the Summer toolkit.

To access Google Colab, you'll need a Google account. If you don't already have one, please take a moment to create one before the session. You can sign up for a Google account here.

The following Colab Notebooks illustrate various epidemiological processes using the summer2 platform.

To run these over Google Colab, follow the links below and the instructions in the first cell.

1. Simple SEIR model
2. Stratification
3. Calibration
4. Heterogeneous mixing
5. Serial latent compartments
6. Tracking model outputs
7. Multiple strains
8. Post-immunity dynamics
9. Renewal introduction
10. Renewal implementation
11. Tuberculosis

What are we trying to do with a compartmental model?

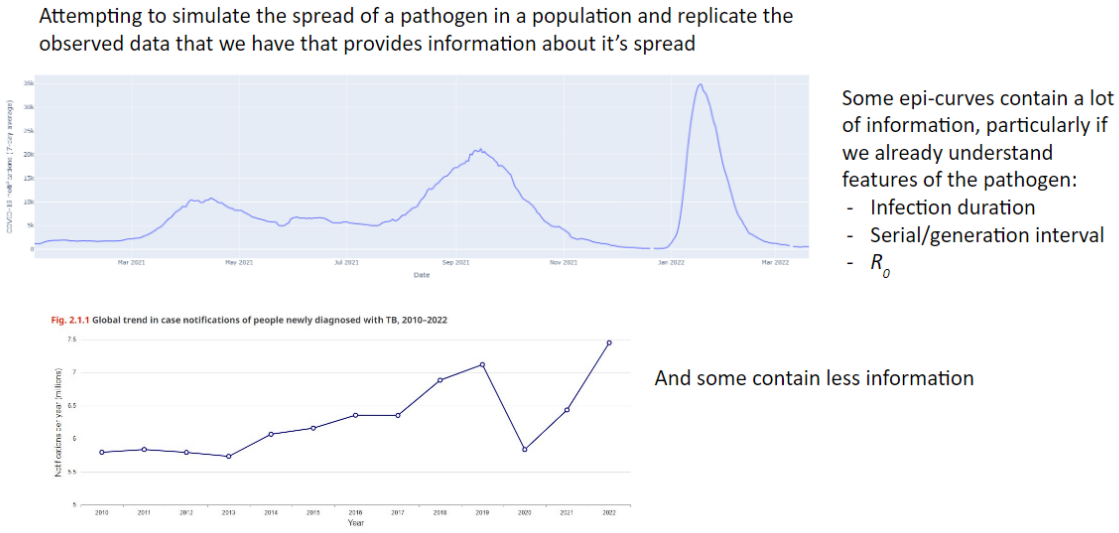


Figure 2: image.png

Figure 2: Simulating the spread of a pathogen involves creating models that can replicate observed data, which helps in understanding and predicting the epidemic's course. The amount of information an epi-curve provides can vary. Detailed curves are more informative when key characteristics of the pathogen are known, while less detailed curves can still offer insights into broader trends. Understanding these curves is crucial for public health planning, resource allocation, and implementing effective control measures.

Broad uses of compartmental modelling in infectious diseases

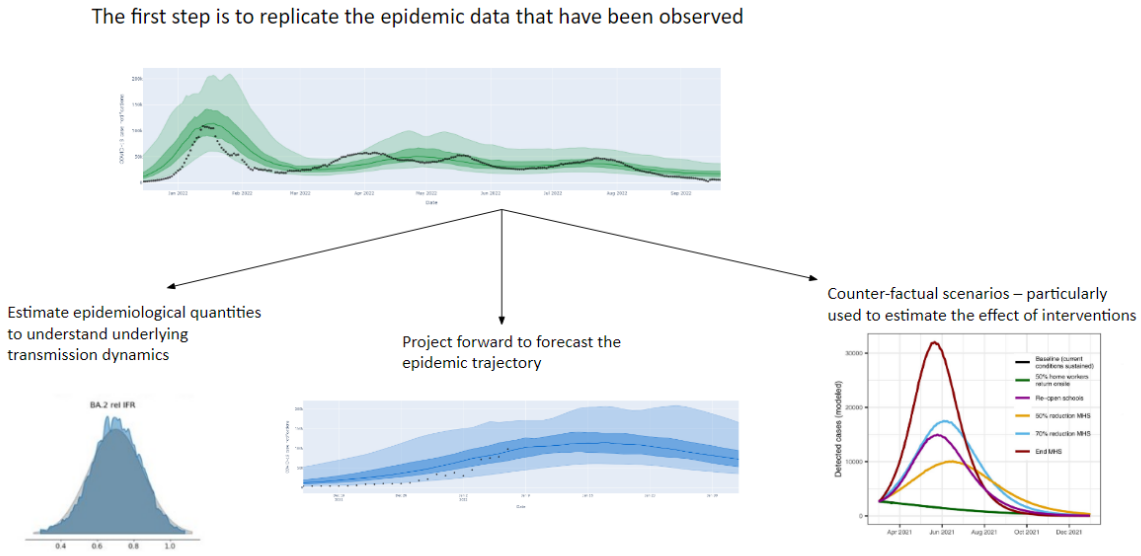


Figure 3: image.png

Figure 3: The first step in epidemic modeling is to replicate the observed data accurately, which provides a foundation for further analysis. Once the model is fitted, it can be used to estimate key epidemiological parameters, project the future course of the epidemic, and evaluate the potential impact of different interventions. This process helps in understanding the epidemic, forecasting future trends, and making informed public health decisions to mitigate the impact of the disease.

What do policy questions look like? Do they always come as questions we can easily model?

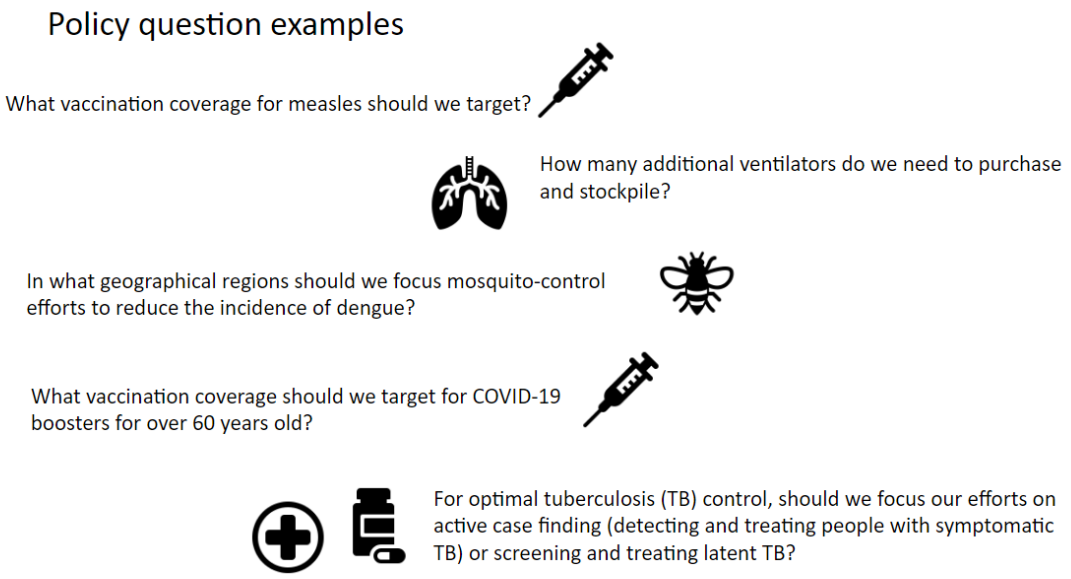


Figure 4: image.png

Can we convert these questions to something we can model?

What vaccination coverage for measles should we target?

The aim of measles vaccination is to achieve sufficient population-level immunity to keep the reproduction number below 1.

Modelling (or epidemiological) question: What is the critical immunity threshold required to keep the effective reproduction below 1?

This question may not need a sophisticated or complex model to answer – but it does require us to go back to first principles for infectious dynamics and modelling, and also basic epidemiology

Critical immunity threshold = 1 – 1/R0, if we assume homogenous population mixing

Some questions we might need consider in order to answer: - What are the estimates for R0 for measles? - How were these R0 derived? Methodology, but also setting/context? - Do we think these estimates are applicable to our context?

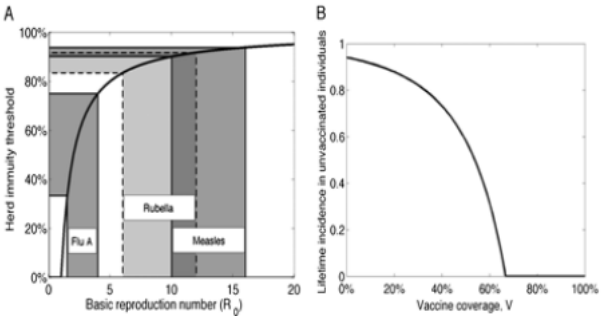


Figure 5: image.png

Fine et al. (2011). "Herd Immunity": A Rough Guide | Clinical Infectious Diseases | Oxford Academic (oup.com)

Figure 4:

Figure A illustrates the relationship between the basic reproduction number (R) and the herd immunity threshold, highlighting the need for higher immunity levels for diseases with higher R .

Figure B demonstrates how increasing vaccine coverage reduces the incidence of disease in unvaccinated individuals, emphasizing the importance of high vaccine uptake to protect the entire population, including those who are not vaccinated.

How many additional ventilators do we need to purchase and stockpile?

This question is a bit more complex -> There is underlying epidemiology (disease natural history) we need to work through in order to convert this to a question we can model that might help answer this question. (we may not be able to answer this question in its entirety)

1. Ventilators are need to provide respiratory support for the most critically unwell cases of COVID-19
2. These cases are captured in critical care (or ICU) admission or occupancy data – in some contexts you may have specific data outlining number of cases on a ventilator
3. Stockpiling is to account for a situation that is at the extreme – interested in the peak of critical care occupancy and under the highest transmission conditions and worst-case scenario assumptions

Modelling (or epidemiological) question: What is the projected size and timing of peak critical care bed occupancy under a situation with no public health or social measures implemented (i.e. population mixing at normal)?

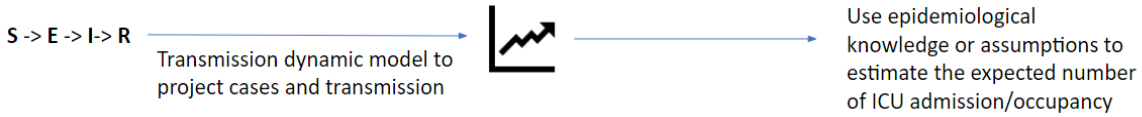


Figure 6: image.png

A transmission dynamic model – might help us understand or explore the theoretical maximum, however the question on how many to stockpile is more complex; Staffing/expertise to use ventilators, ventilator cost, other clinical considerations- i.e. would we put every case on a ventilator

What are our options in responding to a policy question.

Options * Refuse to provide an answer! * Use modelling principles to provide epidemiological context * Identify (partially) applicable modelling analyses from the literature * Use simpler, non-mechanistic approaches * Develop a mechanistic model adapted to the local context

Considerations * Timeframe * Modelling resources / team * Data available

Different model structures to capture different diseases

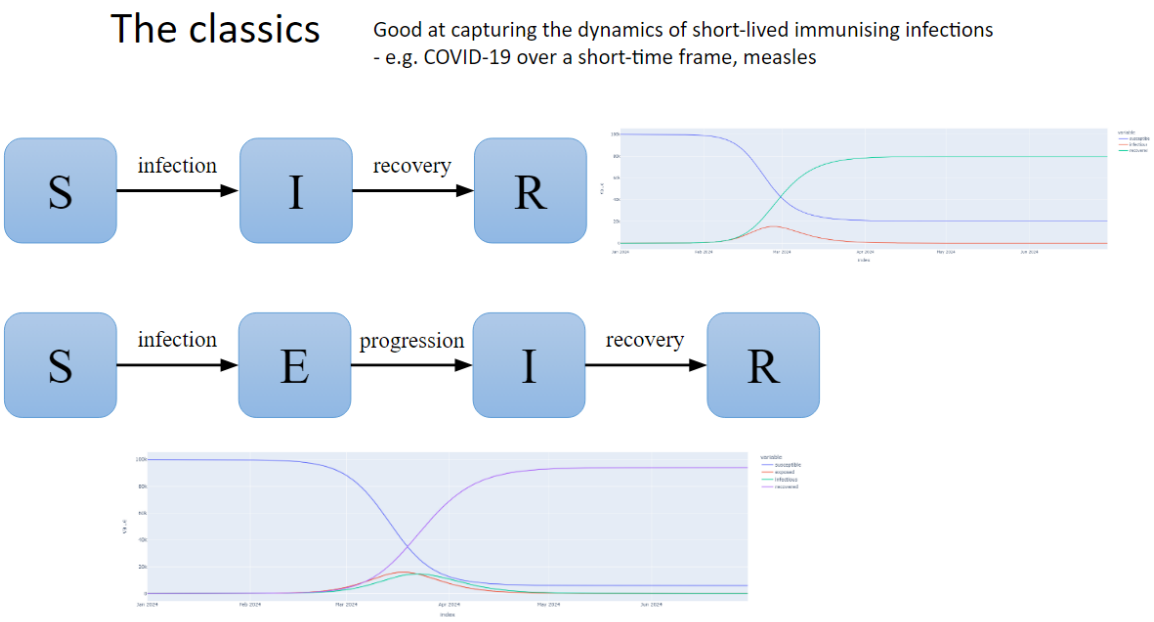


Figure 7: image.png

Figure 5: * **SIR Model:** Useful for diseases with a short incubation period or where the exposed period is negligible. It captures the dynamics between susceptible, infectious, and recovered individuals. * **SEIR Model:** Adds an exposed compartment to account for the incubation period before individuals become infectious. This model is more accurate for diseases with a significant incubation period. * Both models are good at capturing the dynamics of short-lived immunizing infections, providing insights into the progression of the epidemic and the impact of interventions over a short time frame.

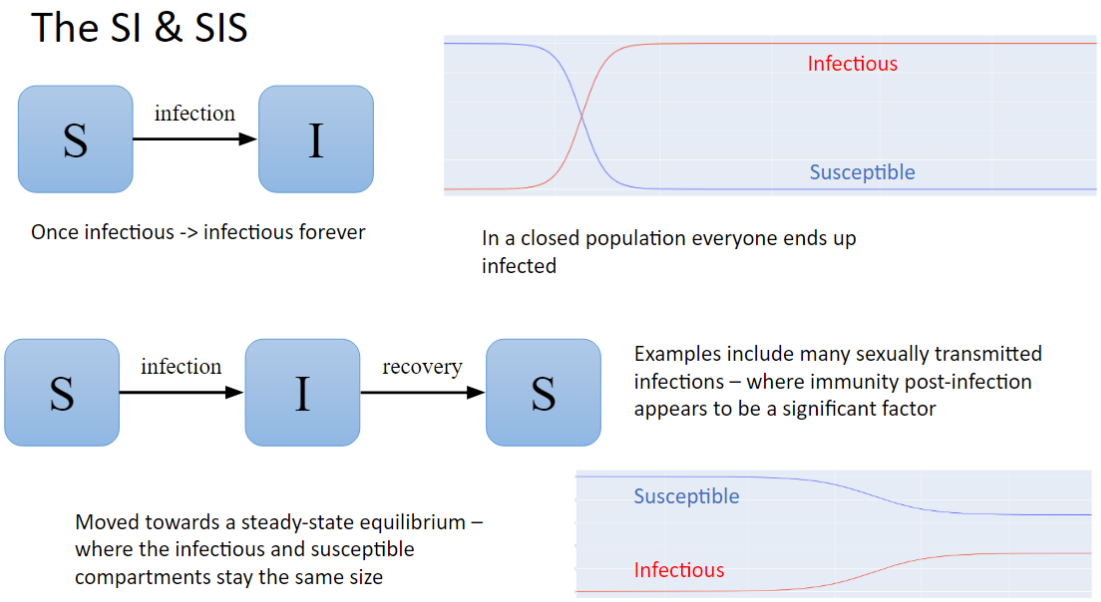


Figure 8: image.png

Figure 6: * **SI Model** Suitable for infections where individuals remain infectious indefinitely. It shows that in a closed population, the infection will eventually spread to everyone. * **SIS Model** Suitable for infections where individuals can recover and become susceptible again. It illustrates a steady-state equilibrium where both susceptible and infectious individuals coexist in the population.

These models help understand the dynamics of diseases with different infection and immunity patterns, providing insights into how they spread and persist in populations.

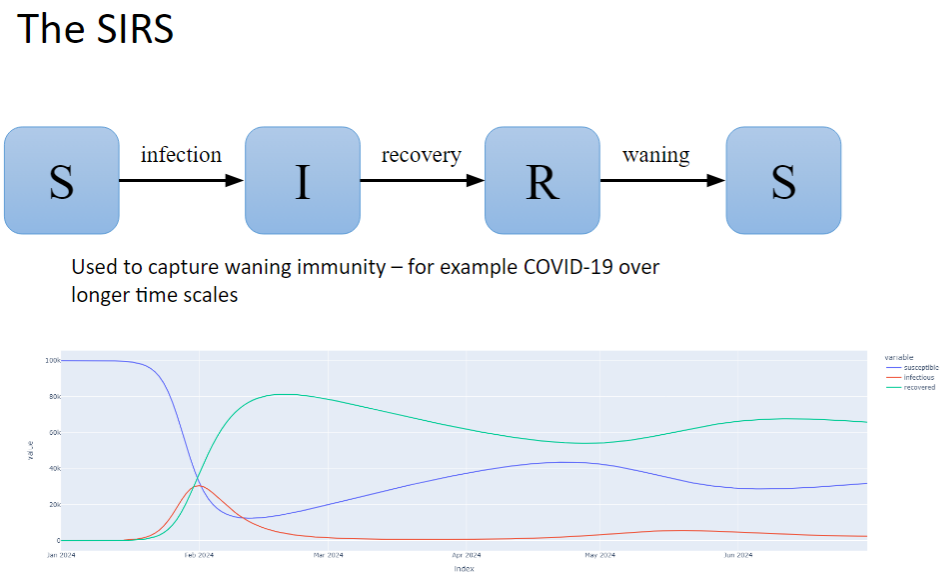


Figure 9: image.png

Figure 7: * The SIRS model captures the dynamics of infections with temporary immunity. * It shows how populations can cycle through stages of susceptibility, infection, and recovery. * The model is valuable for understanding the long-term behavior of diseases like COVID-19, where waning immunity plays a significant role.

This model helps public health officials and researchers predict and manage the potential resurgence of infections and design strategies to mitigate the impact of waning immunity.

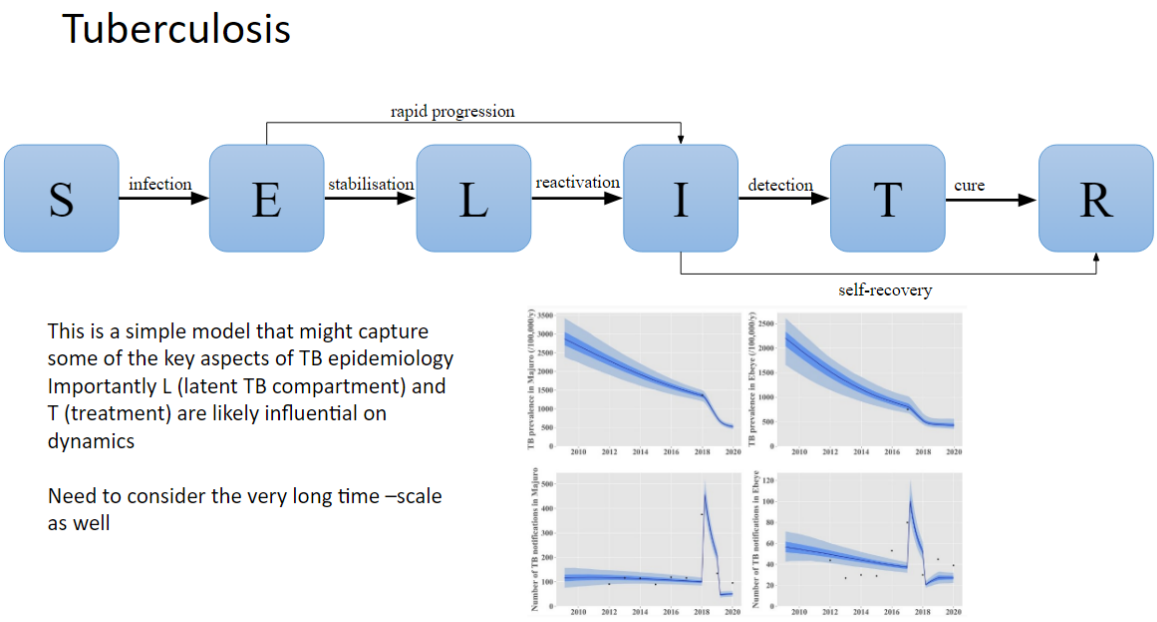


Figure 10: image.png

Figure 8: The SELITR model captures the complex progression of TB, from susceptibility and exposure to latent infection, infectious disease, treatment, and recovery. The model includes key aspects such as latent TB and treatment, which are influential in TB dynamics. Understanding these dynamics over a long time scale is crucial for effective TB control and management strategies.

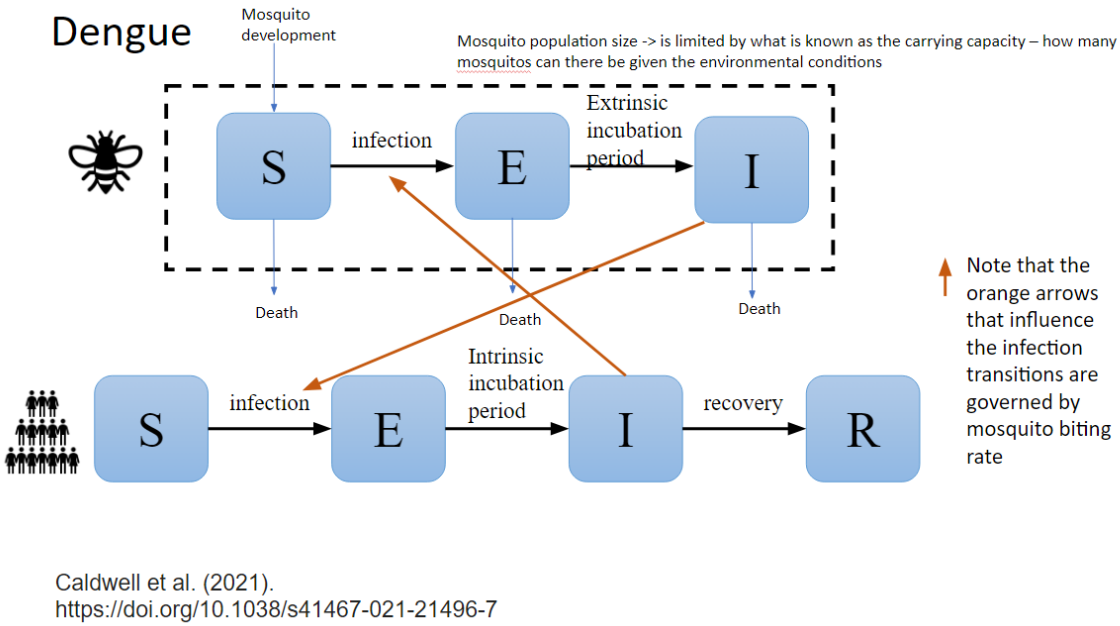


Figure 11: image.png

Figure 9: This Dengue transmission model illustrates the complex interactions between human and mosquito populations, showing how the virus spreads through these hosts. It highlights critical factors such as mosquito biting rates, incubation periods, and population carrying capacity. Understanding these dynamics is essential for designing effective control and prevention strategies for Dengue fever.

Simple SEIR model

The SEIR (Susceptible-Exposed-Infectious-Recovered) model is a widely used epidemiological model that helps us understand the dynamics of infectious diseases.

The SEIR model divides a population into four compartments:

- **Susceptible (S)**: Individuals who are susceptible to the disease but have not been infected yet.
- **Exposed (E)**: Individuals who have been exposed to the disease but are not yet infectious. This represents the latent period during which the infection is incubating.

- **Infectious (I)**: Individuals who are currently infected and can transmit the disease to others.

- **Recovered (R)**: Individuals who have recovered from the infection and are immune.

Assumptions:
* **Constant Population Size (N)**: We assume a fixed population size throughout the epidemic.
* **Constant Rates**: Transmission and removal rates (recovery or death rates) remain constant.
* **No Demography**: Births and deaths are not considered in this simplified model.
* **Well-Mixed Population**: We assume that any infected individual has a probability of contacting any susceptible individual that is reasonably well approximated by the average. This assumption simplifies the model but may not hold in all real-world scenarios.

Epidemic Curve

The figure shows a typical epidemic curve with the following phases:

Exponential Growth Phase:

- At the beginning of an outbreak, the number of infectious individuals grows exponentially.
- This phase is characterized by a rapid increase in the number of cases as susceptible individuals become exposed and then infectious.

Herd Immunity:

- As the epidemic progresses, a significant portion of the population becomes infected and then recovers, gaining immunity.
- Herd immunity is achieved when enough individuals are immune, reducing the overall transmission of the disease.
- This leads to a peak in the epidemic curve, where the number of new cases starts to decline because there are fewer susceptible individuals to infect.

Depletion of Susceptibles:

- As more people become immune, the number of susceptible individuals decreases.
- The epidemic curve declines as the disease has fewer individuals to infect.
- Eventually, the epidemic subsides as the susceptible population is depleted, and most individuals are either recovered or immune.

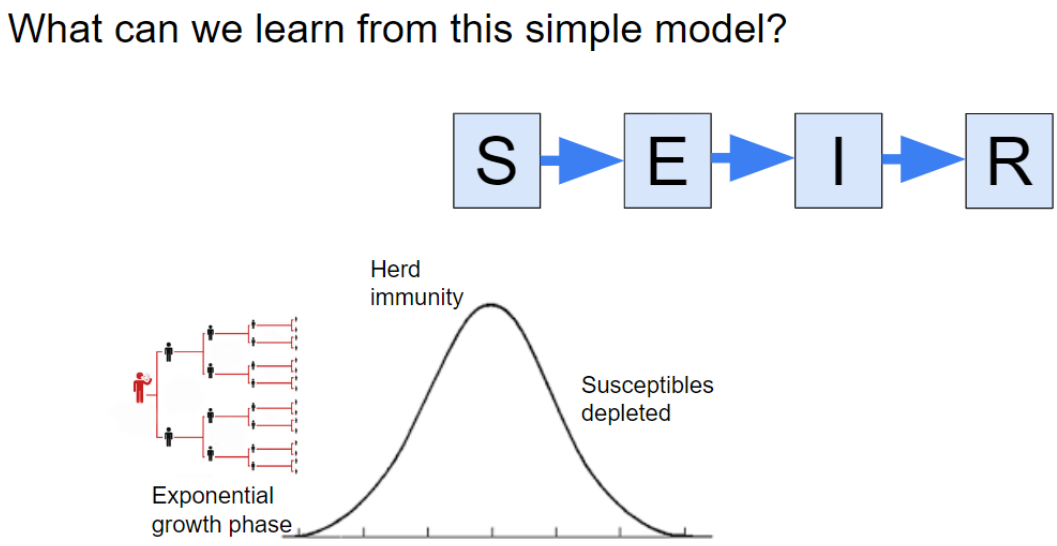


Figure 12: image.png

Figure 10: The SEIR model offers valuable insights into the dynamics of infectious diseases, highlighting the exponential growth phase, the achievement of herd immunity, and the depletion of susceptibles. It underscores the importance of timely public health interventions and helps predict and manage the course of epidemics effectively.

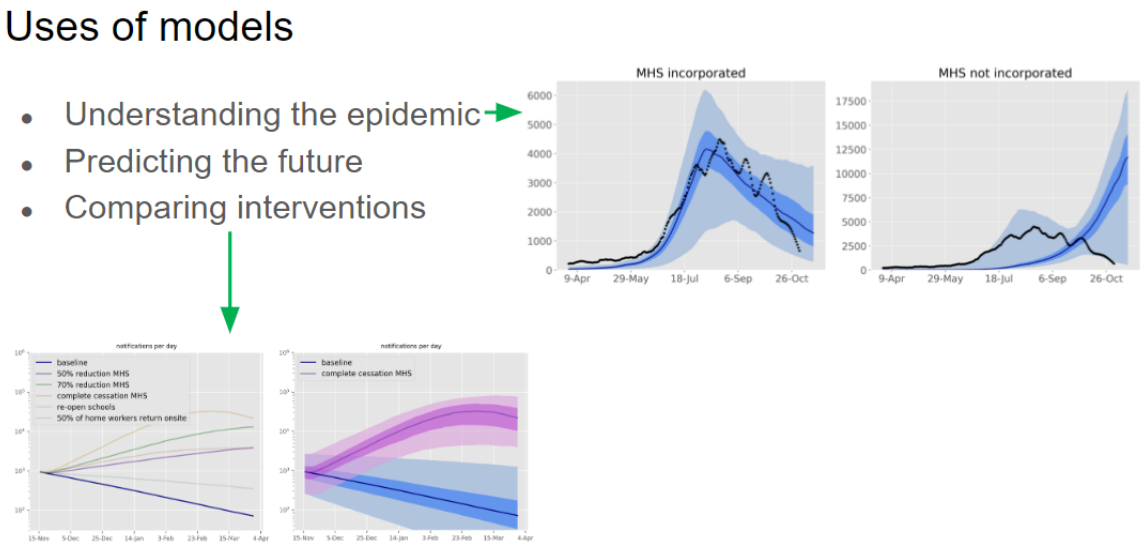


Figure 13: image.png

Figure 11: Overall, epidemiological models are powerful tools for making informed decisions during an epidemic, enhancing our ability to respond effectively and mitigate the impact of infectious diseases on populations.

Uses of Models in Epidemiology:

- **Understanding the Epidemic:** Models help us understand the dynamics of an epidemic, including how it spreads and the impact of various factors on its trajectory.
- **Predicting the Future:** Predictive models provide forecasts of future trends, helping public health officials and policymakers plan and prepare for different scenarios.
- **Comparing Interventions:** Models allow us to compare the effectiveness of various public health interventions, guiding decisions on the best strategies to implement for controlling the epidemic.

The following exercise will provide you the fundamentals for developing a simple SEIR model.

The below code snippet is a conditional installation instruction for the summerepi2 library, specifically tailored for Google Colab environments:

If running from Colab, please uncomment and run the following cell to get summer installed.: This is a comment that provides instructions to the user. It indicates that the following line of

code should be uncommented (by removing the # symbol) and executed if the code is being run within a Google Colab environment.

!pip install summerepi2==1.3.5: This line uses the !pip command to install the summerepi2 library. The ==1.3.5 specifies that version 1.3.5 of the library should be installed. This ensures that the code uses a specific version, avoiding potential compatibility issues that might arise from using different versions.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install summerepi2==1.3.5
```

This code snippet imports necessary libraries and sets up configurations for data analysis and modeling:

1. import pandas as pd: Imports the pandas library and assigns it the alias pd. Pandas is a powerful tool for data manipulation and analysis in Python.
2. from datetime import datetime: Imports the datetime class from the datetime module. This class is used for working with dates and times in Python.
3. from summer2 import CompartmentalModel: Imports the CompartmentalModel class from the summer2 library. This class is likely used for creating compartmental models, which are commonly used in epidemiology and other fields to simulate the flow of individuals between different states.
4. from summer2.parameters import Parameter: Imports the Parameter class from the parameters module within the summer2 library. This class is likely used for defining parameters used in the compartmental models.

```
import pandas as pd
from datetime import datetime
from summer2 import CompartmentalModel
from summer2.parameters import Parameter
```

Model construction and execution

The following cells constitute the epidemiological analysis of this notebook. The code is intended to be very simple and adaptable. Please feel free to adjust any of the data sources or parameters, which is the purpose of this notebook.

Country selection

First choose your country using its ISO code, with options only available for Malaysia, the Philippines and Vietnam. This will determine the Our World in Data (OWID) case data that can be used to compare your model results against. You can use our approximate estimate of the total population of the country, or replace the value as per your preference.

Dates

Feel free to edit the datetime objects that specify the start and end dates for the analysis, the reference ("zero") date and the left limit for the x-axis in the final plot.

Model

The model is a parameterised summer2 SEIR model with frequency-dependent transmission and partial observation of incidence (termed notifications).

Results

The SEIR model provides a poor-to-moderate fit to the target data for the Omicron waves in each of the three countries. Please adjust parameters and dates to achieve a better fit for your country.

```
iso = 'MYS'
```

```
cases_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_cases.csv',
 ↪  index_col=0)[iso]
cases_data.index = pd.to_datetime(cases_data.index)
approx_pops = {
    'MYS': 33e6,
    'PHL': 114e6,
    'VNM': 97e6,
}
```

```
analysis_start_date = datetime(2021, 8, 1)
analysis_end_date = datetime(2022, 6, 1)
epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
```

```
    ['susceptible', 'exposed', 'infectious', 'recovered'],
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'infectious': 1.0})
incidence = epi_model.request_output_for_flow('incidence', 'progression',
 ↪  save_results=False)
epi_model.request_function_output('notifications', incidence *
 ↪  Parameter('detection_prop'));
```

```
parameters = {
    'contact_rate': 0.4,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.07,
}
epi_model.run(parameters)
```

This following code snippet generates and customizes a plot to compare modeled epidemiological data with reported case data:

1. **plot_start_date = datetime(2021, 10, 1)**: Sets the starting date for the plot to October 1, 2021, using the datetime object. This indicates that the plot will display data from this date onwards.
2. **comparison_df = pd.concat([epi_model.get_derived_outputs_df(), cases_data])**: Concatenates (combines) two dataframes: the derived outputs from the epidemiological model (epi_model.get_derived_outputs_df()) and the reported case data (cases_data). This creates a single dataframe containing both modeled and reported data for comparison.
3. **comparison_df.columns = ['modelled', 'reported']**: Renames the columns of the combined dataframe to 'modelled' and 'reported' for clarity and easy reference.

4. **comparison_plot = comparison_df.plot()**: Generates a plot using the combined dataframe. This plot likely visualizes the trends in both modeled and reported data over time.

5. **comparison_plot.update_xaxes(range=(plot_start_date, analysis_end_date))**: Adjusts the x-axis (time axis) of the plot to display data within the range from plot_start_date (October 1, 2021) to analysis_end_date (which is assumed to be defined elsewhere in the code). This focuses the plot on a specific time period for analysis.

In summary, this code prepares data from an epidemiological model and reported cases, combines them, generates a plot comparing the two datasets, and customizes the plot's x-axis to display a specific time range. This visualization aids in assessing the model's performance by comparing its predictions with actual reported data.

```
plot_start_date = datetime(2021, 10, 1)
comparison_df = pd.concat([epi_model.get_derived_outputs_df(), cases_data])
comparison_df.columns = ['modelled', 'reported']
comparison_df.plot(xlim=(plot_start_date, analysis_end_date))
```

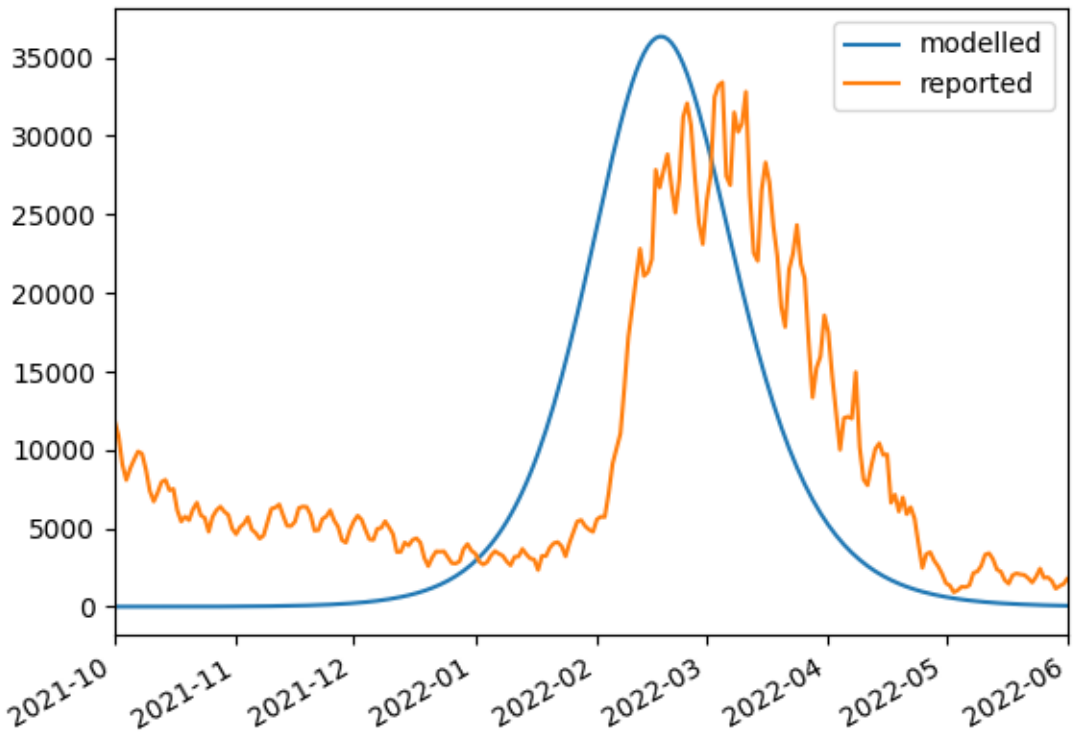


Figure 12: This visualisation aids in assessing the accuracy and reliability of the epidemiological model by comparing its predictions with actual reported data. Identifying discrepancies

between the modelled and reported cases helps in refining the model, making it a crucial tool for public health planning and intervention strategies.

Comparison and Evaluation:

- By comparing the two lines, we can assess how well the model captures the overall pattern of the epidemic.
- Discrepancies between the modelled and reported cases may indicate areas where the model could be improved or adjusted to better reflect the observed data.

Stratification

Stratification in epidemiological models is the process of dividing a population into subgroups or strata based on specific characteristics, such as age, sex, location, or risk factors.

Reasoning: This is done to account for differences in disease transmission and progression within these subgroups. By modeling each stratum separately, you can capture the heterogeneity of the population and obtain more accurate and relevant results.

Here are some benefits of stratification:

Improved accuracy: Captures variations in disease dynamics among different population groups.
Targeted interventions: Helps design and evaluate interventions tailored to specific subgroups.
Better understanding of disease patterns: Reveals how disease spreads and affects different segments of the population.

For example, in a COVID-19 model, you might stratify by age to account for differences in susceptibility and severity of the disease between children, adults, and the elderly.

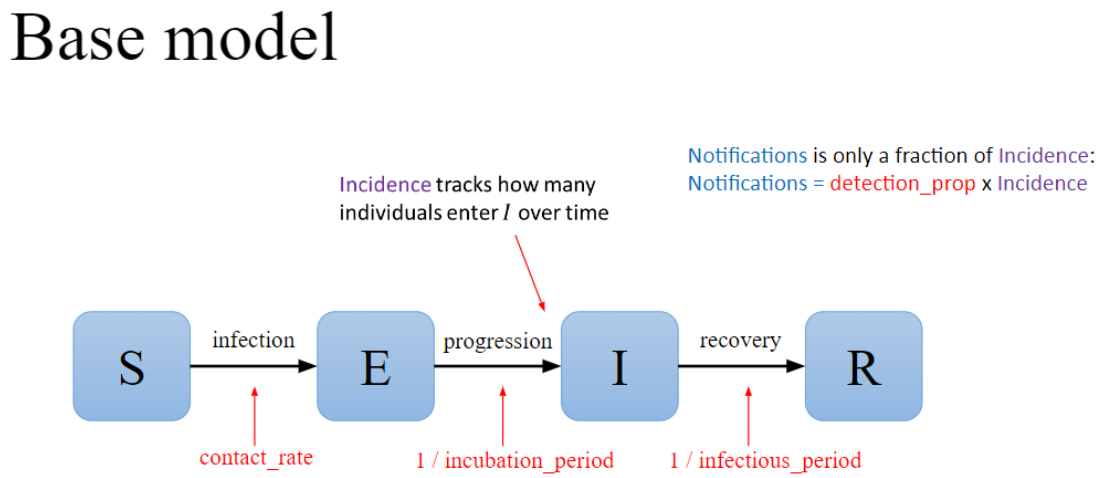


Figure 14: image.png

Figure 13: The SEIR model helps to understand and simulate the spread of infectious diseases through a population by considering the progression through susceptible, exposed, infectious, and recovered stages. It provides insights into the dynamics of infection and recovery, and how public health interventions can impact the course of an epidemic. Notifications and incidence rates are crucial for tracking the disease and implementing effective control measures.

Why stratify a model?

- We may want to represent different population subgroups explicitly with our model. This could be because:
- We identified factors that have a significant influence on the disease of interest (e.g. age, vaccination status, location…).
- We model an intervention that only applies to a specific subgroup (e.g. TB preventive treatment for children, COVID-19 antiviral treatment in older individuals).
- Assuming that, within the modelled population, everyone interacts with everyone at the same rate may not be realistic.
- We want to report information on a specific subgroup (e.g. COVID-19 notifications in 0-15 years old).

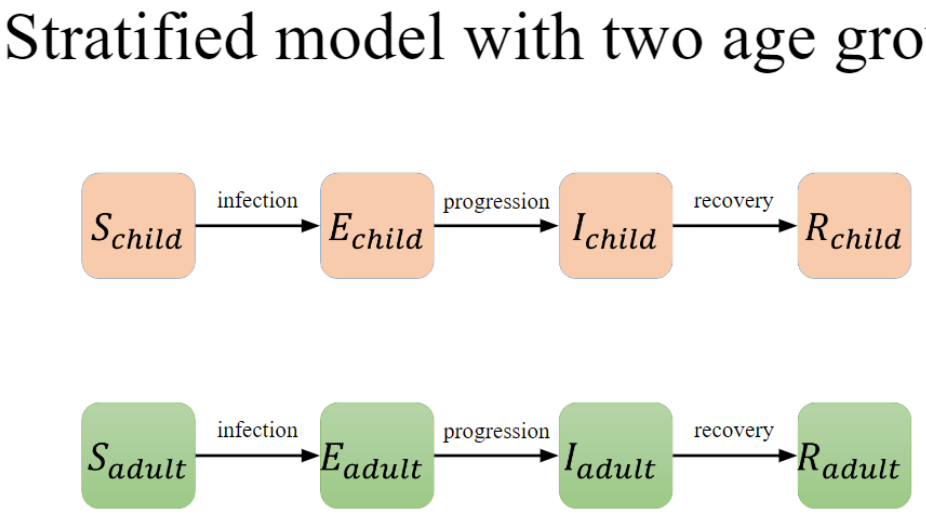


Figure 15: image.png

Figure 14: The stratified SEIR model with two age groups provides a more detailed and accurate representation of the epidemic dynamics by accounting for age-specific characteristics. It allows for better planning and implementation of targeted interventions, ultimately leading to more effective control and mitigation of the disease spread in different segments of the population.

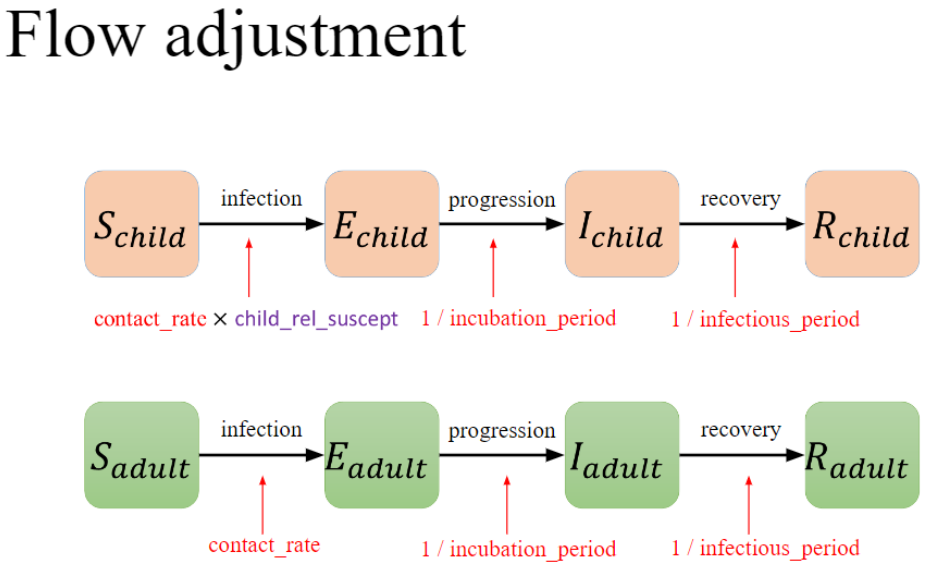


Figure 16: image.png

Figure 15: The adjusted stratified SEIR model enhances the basic stratified model by incorporating age-specific susceptibility adjustments. This refinement allows for more accurate simulations of disease dynamics, reflecting real-world scenarios where different age groups may have varying risks of infection. This improved model is especially useful for tailoring public health interventions and understanding the spread of diseases in diverse demographic settings.

Age-Specific Infection Rate Adjustment:

- The model accounts for the fact that children and adults might have different susceptibilities to infection. This is represented by the term child_rel_suscept child_rel_suscept, which modifies the contact rate for children.
- This adjustment is crucial for accurately modeling the spread of the disease in populations where susceptibility varies by age.

Progression and Recovery Rates:

- Both children and adults follow the same rates for progression from exposed to infectious and from infectious to recovered. This assumes that once exposed, the biological progression of the disease follows similar timelines across age groups.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install summerepi2==1.3.5
```

```
import pandas as pd
from datetime import datetime
from summer2 import CompartmentalModel, Stratification
from summer2.parameters import Parameter
```

Stratification with reduced susceptiblity

Stratification

This notebook applies a simple two-strata stratification to the basic model introduced in the SEIR model notebook. The two strata are arbitrarily referred to as child and adult implying that they are intended to represent age stratification, but this could easily represent other population heterogeneities.

Susceptibility modification

After creating the stratification object, we modify the relative susceptibilty of the "child" stratum. Here we allow for children to be half as susceptible as adults, by modifying the infection flow for children (that is, they get infected at half the rate that they would have been infected if the modification had not been applied - implying that they are half as susceptible to infection).

More generally, this code is intended to represent the way in which users can modify parameter values for subpopulations of the model.

To understand the effect on each population sub-group we will need to track a stratum-specific output.

```
iso = 'MYS'
```

```
cases_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_cases.csv',
 ↪  index_col=0)[iso]
cases_data.index = pd.to_datetime(cases_data.index)
approx_pops = {
    'MYS': 33e6,
```

```
    'PHL': 114e6,
    'VNM': 97e6,
}
```

```
analysis_start_date = datetime(2021, 8, 1)
analysis_end_date = datetime(2022, 6, 1)

epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
    ['susceptible', 'exposed', 'infectious', 'recovered'],
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'infectious': 1.0})
```

This following code snippet demonstrates the process of stratifying a compartmental model, based on age groups:

1. **age_strata = ['child', 'adult']**: Defines a list named age_strata containing the age groups 'child' and 'adult'. These will be used to stratify the model.
2. **age_strat = Stratification('age', age_strata, epi_model._original_compartment_names)**: Creates a Stratification object named age_strat. This object likely represents the stratification strategy based on the 'age' attribute, using the defined age_strata and the original compartment names from an existing epidemiological model (epi_model).
3. **age_suscept = {'child': Parameter('child_rel_suscept'), 'adult': None}**: Creates a dictionary age_suscept to specify relative susceptibilities for each age group. Children have a susceptibility defined by the parameter 'child_rel_suscept', while adults have no specified adjustment (assumed to be the baseline).
4. **age_strat.set_flow_adjustments('infection', age_suscept)**: Applies the age-specific susceptibilities to the 'infection' flow within the stratification strategy. This means the infection rate will be adjusted differently for children and adults based on the provided parameters.

5. **epi_model.stratify_with(age_strat)**: Stratifies the epidemiological model epi_model using the defined age stratification strategy age_strat. This step incorporates the age groups and their specific characteristics into the model.

6. **for strat in age_strata: ...**: This loop iterates through each age stratum ('child' and 'adult') to likely perform further operations or modifications to the model based on each age group. The incomplete code suggests that it might be defining or adjusting incidence rates (inc_string and age_inc) for each stratum.

```
# Stratification
age_strata = ['child', 'adult']
age_strat = Stratification('age', age_strata,
 ↪  epi_model._original_compartment_names)
age_suscept = {'child': Parameter('child_rel_suscept'), 'adult': None}
age_strat.set_flow_adjustments('infection', age_suscept)
epi_model.stratify_with(age_strat)
for strat in age_strata:
    inc_string = f'incX{strat}'
    age_inc = epi_model.request_output_for_flow(inc_string, 'progression',
 ↪  source_strata={'age': strat}, save_results=False)
    epi_model.request_function_output(f'notifX{strat}', age_inc *
 ↪  Parameter('detection_prop'))
```

```
parameters = {
    'contact_rate': 0.5,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.07,
    'child_rel_suscept': 0.5,
}
epi_model.run(parameters)
```

This code snippet creates an area plot to visualize the derived outputs of an epidemiological model, specifically focusing on notifications stratified by age, and overlays it with reported case data as markers:

1. **plot_start_date = datetime(2021, 10, 1)**: Sets the starting date for the plot's x-axis to October 1, 2021.

2. **plot = epi_model.get_derived_outputs_df()[[f'notifX{strat}' for strat in age_strata]].plot.area()**:

- Extracts the derived outputs from the epidemiological model (epi_model.get_derived_outputs_df()).

- Selects columns representing notifications for each age stratum using a list comprehension: [f'notifX{strat}' for strat in age_strata].
- Creates an area plot using the selected columns (plot.area()), where the area under each curve represents the cumulative number of notifications for each age group over time.

3. **plot.update_xaxes(range=(plot_start_date, analysis_end_date))**: Adjusts the x-axis to display the time range from plot_start_date to analysis_end_date.
4. **plot.add_trace(go.Scatter(x=cases_data.index, y=cases_data, mode='markers', name='data'))**:

- Adds a scatter plot trace to the existing area plot.
- Uses the go.Scatter function (likely from the Plotly library) to create a scatter plot with markers.
- Sets the x-values to the index of the cases_data dataframe (likely dates or time points).
- Sets the y-values to the actual reported case data (cases_data).
- Labels the scatter plot trace as 'data'.

In summary, this code generates an area plot showing the cumulative notifications for different age groups over time, based on the epidemiological model. It then overlays the reported case data as markers on top of the area plot, allowing for a direct comparison between the model's predictions and the actual observations. This visualization helps assess the model's ability to capture the dynamics of notifications and their distribution across age groups.

```
plot_start_date = datetime(2021, 10, 1)
plot = epi_model.get_derived_outputs_df()[[f'notifX{strat}' for strat in
 ↪  age_strata]].plot.area(xlim=(plot_start_date, analysis_end_date))
cases_data[plot_start_date:analysis_end_date].plot(style='.', color='black')
```

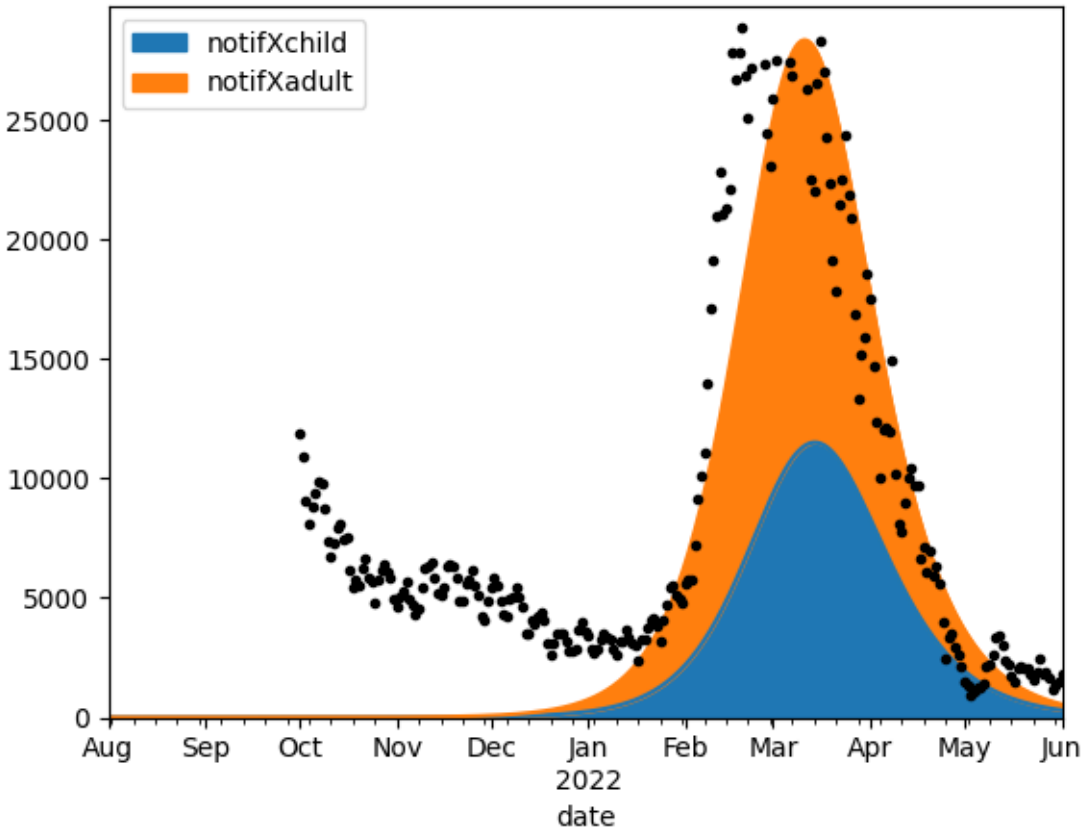


Figure 16: This visualisation aids in understanding the dynamics of disease notifications across different age groups by comparing model predictions with actual reported data. The area plot helps to see the cumulative effect over time, while the overlay of actual data points provides a clear comparison, highlighting how well the model aligns with real-world observations. Such comparisons are crucial for validating and refining epidemiological models to ensure accurate and reliable predictions, which are essential for effective public health planning and intervention strategies.

Evaluating Model Performance:

- By comparing the modelled areas with the actual data markers, we can assess how well the model captures the dynamics of notifications and their distribution across age groups.
- Discrepancies between the model predictions and reported cases may indicate areas where the model could be improved.

Calibration

Calibration in epidemiological modeling is the process of adjusting model parameters to ensure that the model's outputs closely match observed data.

Reasoning: The goal is to create a model that accurately reflects the real-world dynamics of the disease being studied.

Here's why calibration is important:

- **Improved accuracy**: Ensures that the model's predictions are reliable and reflect the actual situation.
- **Increased confidence**: Increases confidence in the model's ability to project future trends and evaluate interventions.
- **Better decision-making**: Provides a more reliable basis for making informed public health decisions.

Common calibration techniques:

- **Manual adjustment**: Experts iteratively adjust parameters based on their knowledge of the disease and the observed data.
- **Optimization algorithms**: Algorithms systematically search for parameter values that minimize the difference between model outputs and observed data.

By calibrating an epidemiological model, you increase its ability to provide valuable insights and support effective public health interventions.

Parameter Space

The concept of parameter space is used to explore and understand how different parameter values affect the outputs of an epidemiological model. This figure illustrates the parameter space for two key parameters: the contact rate and the detection proportion.

Key Components 1. Axes

- X-axis (Contact Rate): Represents the rate at which susceptible individuals come into contact with infectious individuals.
- Y-axis (Detection Proportion): Represents the proportion of actual cases that are detected and reported.

Parameter Space Area

- The green shaded area represents the entire range of possible values for the contact rate and detection proportion.

- Each point within this area corresponds to a specific combination of these two parameters.

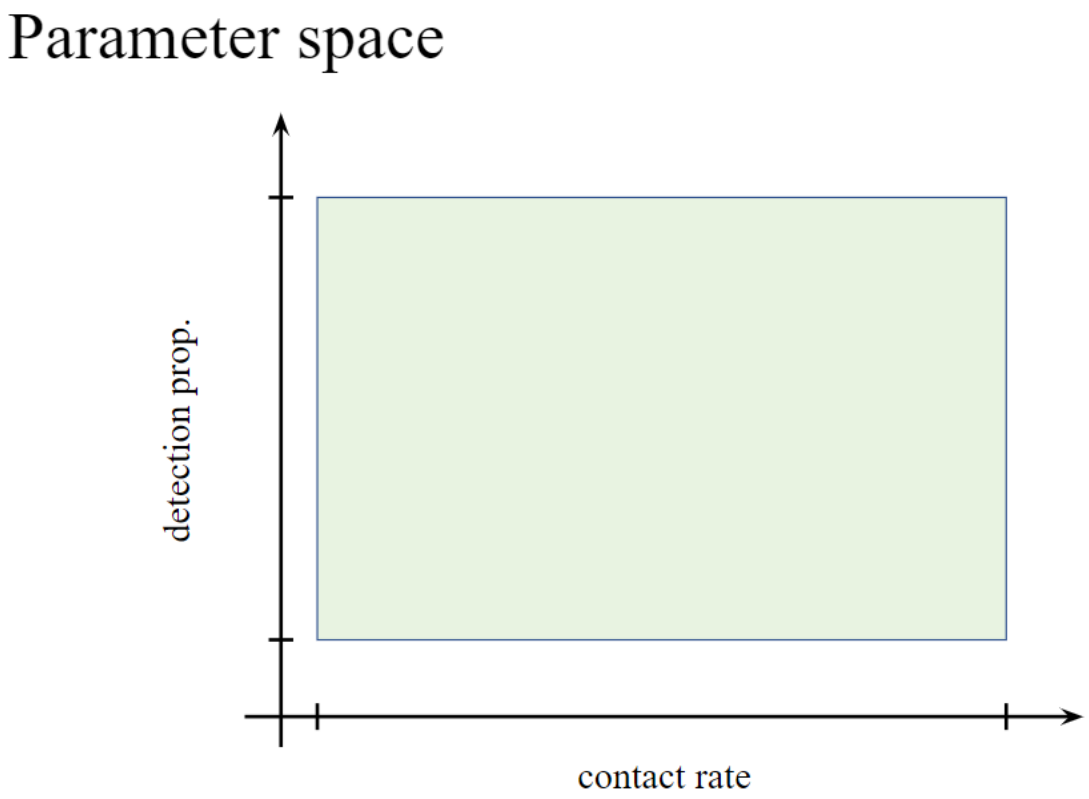


Figure 17: image.png

Figure 17: The parameter space diagram visually represents the range of possible values for key model parameters, specifically the contact rate and detection proportion. Each point in the parameter space corresponds to a unique parameter set, which in turn produces a unique model prediction. Exploring the parameter space is essential for understanding model sensitivity, accurately estimating parameters, and calibrating the model to ensure it accurately reflects real-world dynamics.

Parameter Set

- A parameter set is a specific combination of values for the parameters used in the model.
- Each point in the parameter space corresponds to a different parameter set, and each parameter set produces a unique model output.

Model Sensitivity:

- Exploring different points in the parameter space allows us to understand how sensitive the model is to changes in the contact rate and detection proportion.
- By analyzing multiple parameter sets, we can identify which parameters have the most significant impact on the model's predictions.

Parameter Estimation:

- Accurate estimation of the contact rate and detection proportion is crucial for making reliable predictions.
- The red dot represents a tested or estimated parameter set, but the goal is to find the best-fitting parameter set by comparing model predictions with actual data.

Model Calibration:

- The process of model calibration involves adjusting the parameters within the parameter space to minimize the difference between the model's predictions and the observed data.
- This helps in fine-tuning the model to improve its accuracy and reliability.

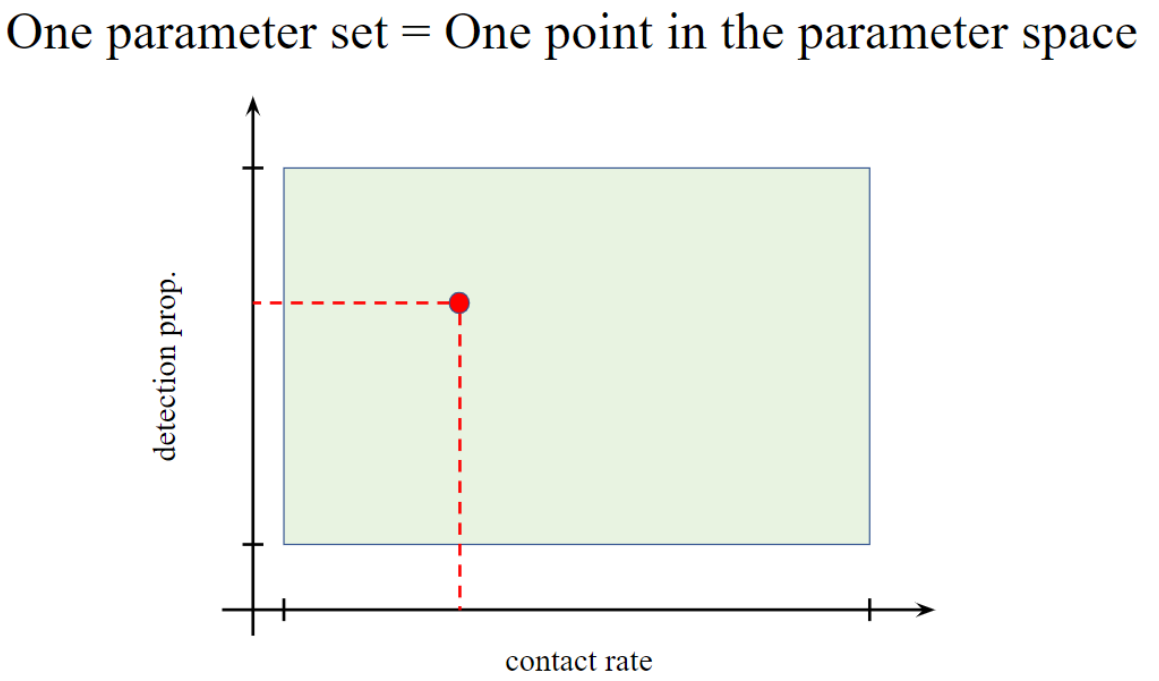


Figure 18: image.png

Figure 18: The parameter space diagram visually represents the range of possible values for key model parameters, specifically the contact rate and detection proportion. Each point in the parameter space corresponds to a unique parameter set, which in turn produces a unique model prediction. Understanding and exploring the parameter space is essential for accurately

estimating parameters, assessing model sensitivity, and calibrating the model to ensure it accurately reflects real-world dynamics.

Interpretation

By varying the contact rate and detection proportion within the parameter space, different scenarios can be modeled to understand the impact of these parameters on the spread of the disease and the reported cases.

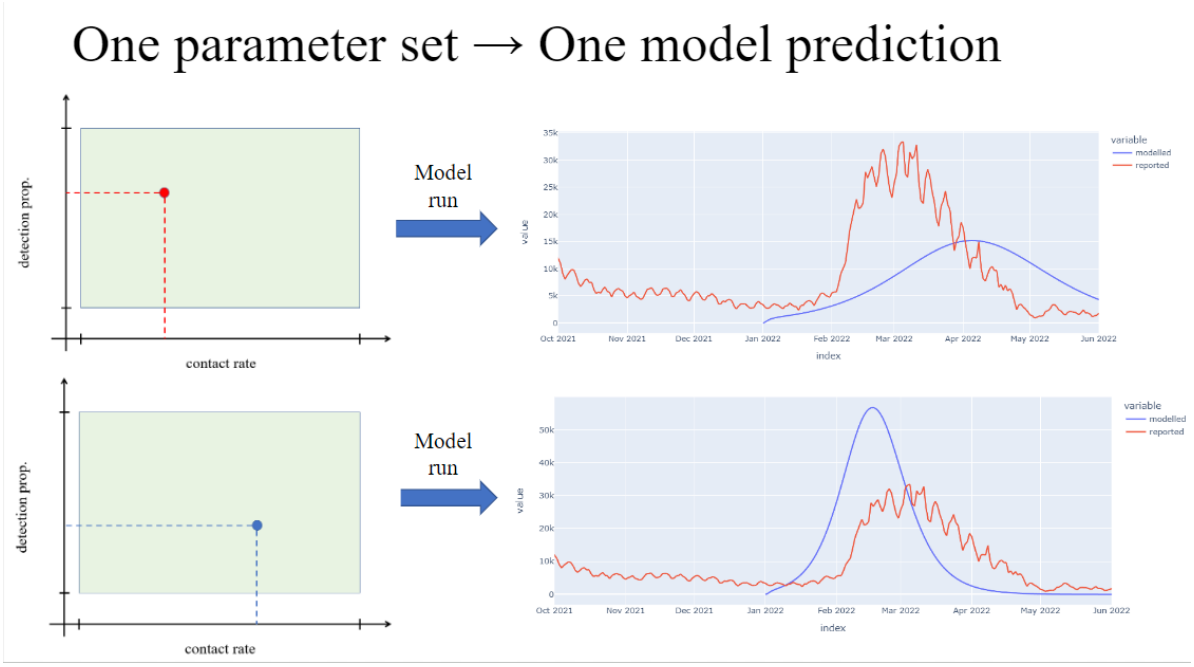


Figure 19: image.png

Figure 19: This figure underscores the critical role of parameter selection in epidemiological modeling. Different parameter sets yield different model predictions, emphasizing the need for accurate and context-specific parameter estimation. Comparing model predictions with actual reported cases helps in evaluating and refining the model, ensuring it accurately reflects real-world dynamics. Understanding the sensitivity of the model to various parameters is crucial for making informed public health decisions and effectively managing infectious disease outbreaks.

One Parameter Set → One Model Prediction

This section shows how different parameter sets lead to different model predictions:

Top Parameter Set (Red Dot):

- The red dot represents one specific set of values for the contact rate and detection proportion.
- Running the model with this parameter set generates a prediction (shown in the top right graph) where the blue line represents the modeled incidence, and the red line represents the reported cases.

Bottom Parameter Set (Blue Dot):

- The blue dot represents a different set of values for the contact rate and detection proportion.
- Running the model with this new parameter set generates a different prediction (shown in the bottom right graph).

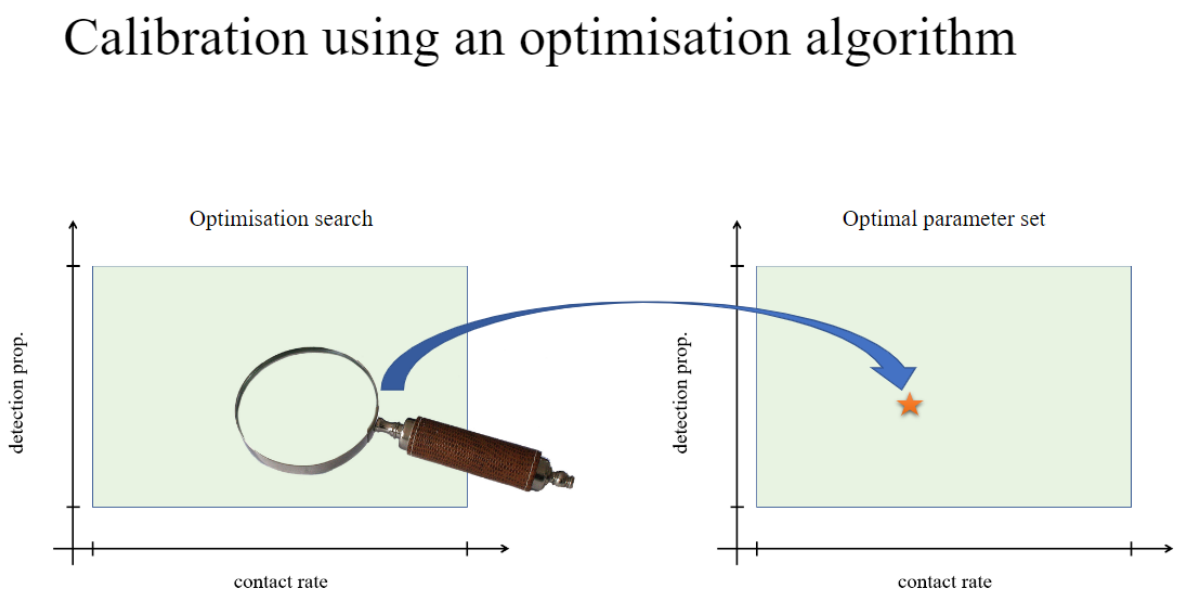


Figure 20: image.png

Figure 20: The figure illustrates the calibration process where an optimization algorithm is used to find the optimal parameter set within the parameter space. This systematic search improves the model's accuracy by fine-tuning the parameters to best fit the observed data. Calibration is a crucial step in epidemiological modeling, ensuring that the model reliably reflects real-world dynamics and provides accurate predictions for public health planning and intervention strategies.

Calibration Using an Optimisation Algorithm

This section demonstrates how an optimization algorithm is used to calibrate the model:

Optimisation Search:

- The process begins with an optimization search in the parameter space (left diagram).
- The magnifying glass symbolizes the search for the best parameter set that aligns the model predictions with the observed data.

Optimal Parameter Set:

- The optimization algorithm searches through various parameter sets to find the optimal one.
- The optimal parameter set (represented by the star in the right diagram) is the one that provides the best fit between the modeled predictions and the observed data.

Summary * One Parameter Set → One Model Prediction: Each specific set of parameter values (contact rate and detection proportion) generates a unique model prediction for the incidence and reported cases. * Calibration: The model calibration process involves using an optimization algorithm to search through the parameter space and identify the parameter set that best fits the observed data, leading to the most accurate model predictions.

This approach ensures that the model is tuned to accurately reflect real-world data, improving the reliability of its predictions for informing public health decisions.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install estival[pymc,nevergrad]==0.5.1 summerepi2==1.3.5
```

```
# Basic imports for summer2 compartmental modelling
import numpy as np
import pandas as pd
from datetime import datetime
from summer2 import CompartmentalModel
from summer2.parameters import Parameter
```

Model construction and execution

The following cells constitute the epidemiological analysis of this notebook. The code is intended to be very simple and adaptable. Please feel free to adjust any of the data sources or parameters, which is the purpose of this notebook.

Country selection

First choose your country using its ISO code, with options only available for Malaysia, the Philippines and Vietnam. This will determine the Our World in Data (OWID) case data that can be used to compare your model results against. You can use our approximate estimate of the total population of the country, or replace the value as per your preference.

Dates

Feel free to edit the datetime objects that specify the start and end dates for the analysis, the reference ("zero") date and the left limit for the x-axis in the final plot.

Model

The model is a parameterised summer2 SEIR model with frequency-dependent transmission and partial observation of incidence (termed notifications).

Results

The SEIR model provides a poor-to-moderate fit to the target data for the Omicron waves in each of the three countries. Please adjust parameters and dates to achieve a better fit for your country.

```
iso = 'MYS'
```

```
cases_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_cases.csv',
 ↪  index_col=0)[iso]
cases_data.index = pd.to_datetime(cases_data.index)
approx_pops = {
    'MYS': 33e6,
    'PHL': 114e6,
    'VNM': 97e6,
}
```

```
analysis_start_date = datetime(2022, 1, 1)
analysis_end_date = datetime(2022, 6, 1)
epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
```

```
    ['susceptible', 'exposed', 'infectious', 'recovered'],
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'infectious': Parameter('initial_infected')})
incidence = epi_model.request_output_for_flow('incidence', 'progression',
 ↪  save_results=False)
epi_model.request_function_output('notifications', incidence *
 ↪  Parameter('detection_prop'));
```

```
parameters = {
    'contact_rate': 0.5,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.07,
    'initial_infected': 100000.0,
}
epi_model.run(parameters)
```

```
plot_start_date = datetime(2021, 10, 1)
comparison_df = pd.concat([epi_model.get_derived_outputs_df(), cases_data])
comparison_df.columns = ['modelled', 'reported']
comparison_plot = comparison_df.plot(xlim=(plot_start_date,
 ↪  analysis_end_date))
```

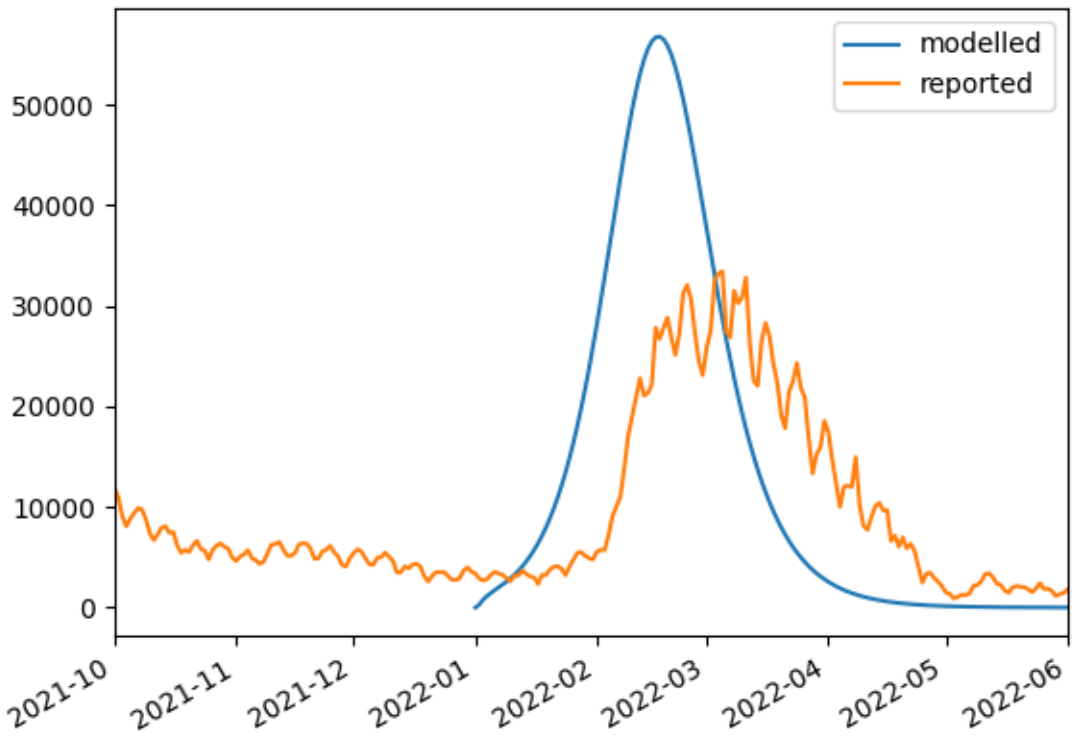


Figure 21: Before calibration

Model calibration

Estival is a Python framework designed for calibrating and optimizing epidemiological models.

Reasoning: It provides tools to connect models to observed data, specify prior assumptions about parameters, and systematically estimate parameter values that best fit the data.

Key features of Estival:

- **Flexible model integration**: Works with various types of epidemiological models.
- **Bayesian parameter estimation**: Allows for incorporating prior knowledge and uncertainty into parameter estimation.
- **Efficient optimization algorithms**: Uses advanced algorithms to find optimal parameter values.
- **User-friendly interface**: Provides a straightforward way to define models, targets, and priors.

By using Estival, modelers can improve the accuracy and reliability of their epidemiological models, leading to better insights and more informed decision-making.

Import calibration tools and create calibration functions

The following code sets the stage for using Estival to calibrate an epidemiological model. It introduces the concepts of targets (observed data) and priors (parameter assumptions) as key building blocks for constructing models within the Estival framework. By connecting the model to data and incorporating prior knowledge, Estival enables a systematic and informed approach to parameter estimation and model optimization.

```
# Estival is our calibration/optimization framework - for connecting models
 ↪  and parameters to data
# The following imports are the 'building blocks' of estival models

# Targets represent data we are trying to fit to
from estival import targets as est

# We specify parameters using (Bayesian) priors
from estival import priors as esp

# Finally we combine these with our summer2 model in a
 ↪  BayesianCompartmentalModel (BCM)
from estival.model import BayesianCompartmentalModel
```

```
WARNING (pytensor.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

```
# import nevergrad tool using estival. nevergrad is a python package for
 ↪  optimisation
from estival.wrappers.nevergrad import optimize_model

# PyMC imports - PyMC is a widely used probabilistic programming framework
 ↪  for Python
from estival.wrappers import pymc as epm
import pymc as pm

# This is required for pymc parallel evaluation in notebooks
import multiprocessing as mp
import platform
```

```
if platform.system() != "Windows":
    mp.set_start_method('forkserver', True)
```

This below code defines the function get_calibration_model that creates a calibration model object using the Estival package. This function facilitates the process of calibrating an epidemiological model against observed data.

1. **def get_calibration_model(model, data):**: Defines a function that takes two arguments:

- **model**: The epidemiological model to be calibrated.
- – : The observed data used for calibration.

2. **“ “ ” ... “ “ “**: This docstring provides a brief description of the function's purpose, arguments, and return value.
3. **# Describe a ‘target’; some observed data against which the model will be ...**: This incomplete comment indicates that the next step would involve defining a “target” within the Estival framework. A target represents the observed data that the model will be calibrated against.

Reasoning: By creating a calibration model object using Estival, this function sets up the framework for connecting the epidemiological model to observed data and systematically adjusting model parameters to achieve a better fit. This calibration process enhances the accuracy and reliability of the model's predictions.

The subsequent steps within the function define targets, specifying priors for model parameters, and configuring the optimization process to find the best-fitting parameter values.

```
def get_calibration_model(model, data):
    """
    Creates a calibration model object using the estival package. This object
 ↪  belongs to the BayesianCompartmentalModel class of estival.

    Args:
        model: the model we are calibrating
        data: the data used for calibration
    """

    # Describe a 'target'; some observed data against which the model will be
     ↪  evaluated
    # (and a description of how this evaluation will be performed)
    targets = [
```

```
        est.TruncatedNormalTarget("notifications", data, (0.0,np.inf), 3000.)
    ]

    # Describe priors for the calibrated parameters - the ranges (and
     ↪ statistical distributions) of what we believe
    # the parameters might (or should) be
    priors = [
        esp.UniformPrior("contact_rate", (0.01,1.0)),
        esp.UniformPrior("detection_prop", (0.01,0.5)),
    ]

    fixed_parameters = {
        'incubation_period': 5.0,
        'infectious_period': 5.0,
        'initial_infected': 100000.0,
    }

    # The BayesianCompartmentalModel class is the primary entry point to all
     ↪ optimization and calibration
    # methods in estival
    # It takes a CompartmentalModel object, default parameters, priors, and
     ↪ targets
    # The default parameters will be used as fixed values when no prior is
     ↪ specified for a given parameter
    bcm = BayesianCompartmentalModel(model, fixed_parameters, priors,
↪   targets)

    return bcm


def calibrate_model_with_optimisation(bcm):
    """
    This function performs a model calibration using optimisation.
    Calibration is performed using the estival package, which implements a
↪   wrapper for optimisation methods provided by the nevergrad package.

    Args:
        bcm: the calibration model object (type BayesianCompartmentalModel)
    """
    # create a nevergrad optimisation runner
    from nevergrad.optimization.differentialevolution import TwoPointsDE
    orunner = optimize_model(bcm, opt_class=TwoPointsDE, num_workers=4,
↪   budget=4000)
```

```
    # perform optimisation, allowing for up to 1000 model evaluations
    rec = orunner.minimize(4000)
    # retrieve optimised parameter values
    optimised_params = rec.value[1]

    return optimised_params


def calibrate_model_with_sampling(bcm):
    """
    This function performs a model calibration using Bayesian sampling.
    Calibration is performed using the estival package, which implements a
 ↪  wrapper for sampling methods provided by the PyMC package.

    Args:
        bcm: the calibration model object (type BayesianCompartmentalModel)
    """

    with pm.Model() as model:
        # This is all you need - a single call to use_model
        variables = epm.use_model(bcm)

        # Now call a sampler using the variables from use_model
        # In this case we use the Differential Evolution Metropolis(Z)
         ↪  sampler
        # See the PyMC docs for more details
        idata = pm.sample(step=[pm.DEMetropolisZ(variables)], draws=2000,
 ↪  tune=2000,cores=4,chains=4)

    return idata
```

Create a calibration model object, which encapsulates our model and the data used for calibration

By creating a calibration model object, you establish a clear framework for improving your model's ability to capture and predict the spread of the disease being studied.

- **Calibration Model Object**: A specialized object that brings together the epidemiological model you've developed and the real-world data you want to use for calibration. Think of it as a container that holds both the model's structure and the data it needs to learn from.

- **Encapsulation**: The calibration model object encapsulates, or bundles together, the model and the data, creating a unified entity for the calibration process. This helps organize and streamline the calibration workflow.
- **Purpose**: The primary purpose of this object is to facilitate the adjustment of model parameters so that the model's outputs align closely with the observed data.

Key benefits of creating a calibration model object:

- **Organized Workflow**: Provides a structured approach to calibration.
- **Efficient Parameter Estimation**: Enables systematic and efficient estimation of model parameters.
- **Improved Model Accuracy**: Leads to a more accurate and reliable model that better reflects real-world dynamics.

```
# Select a subset of the data which will be used for model evaluation during
 ↪  calibration
cases_data_select = cases_data["jan 2022":"may 2022"]
cases_data_select.plot()
```

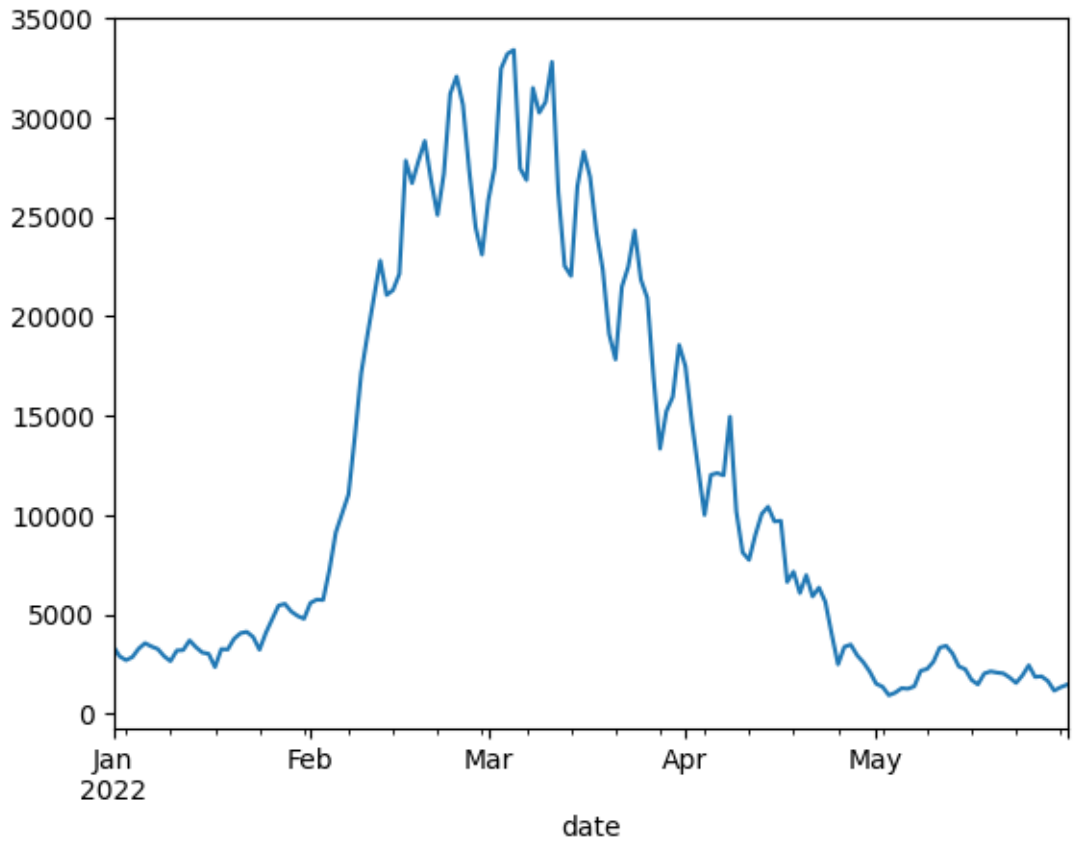


Figure 22: Creating a calibration model object provides a clear framework for systematically calibrating an epidemiological model. This process involves encapsulating the model and the data, facilitating efficient parameter estimation, and improving the model's accuracy. Using a structured approach to select and visualize subsets of data helps in aligning the model's predictions with real-world observations, enhancing its reliability and predictive power.

```
bcm = get_calibration_model(model=epi_model, data=cases_data_select)
```

Perform calibration using parameter optimisation

By performing calibration using parameter optimization, you leverage computational power to systematically fine-tune your model, making it a more powerful tool for understanding and responding to epidemiological challenges.

Parameter Optimization: A systematic approach to finding the best possible values for the model's parameters. It involves searching through a range of parameter values and evaluating how well each set of values allows the model to reproduce the observed data.

How it works: 1. Define an objective function: A measure of how well the model's outputs match the observed data. 2. Employ an optimization algorithm: Algorithms like gradient descent or evolutionary algorithms are used to iteratively adjust the parameters and find the values that minimize the objective function (i.e., maximize the agreement between the model and the data).

Key benefits of parameter optimization for calibration:

- **Systematic Approach**: Replaces manual, trial-and-error parameter adjustments with a more rigorous and efficient method.
- **Optimal Parameter Values**: Identifies parameter values that lead to the best possible fit between the model and the observed data.
- **Enhanced Model Accuracy**: Results in a more accurate and reliable model that can be used for forecasting and evaluating interventions.

```
optimised_params = calibrate_model_with_optimisation(bcm)
optimised_params
```

```
{'contact_rate': 0.3889705476070502, 'detection_prop': 0.06411107441716091}
```

```
# run the modle with the optimised parameter set
res = bcm.run(optimised_params)
```

```
# plot optimal model fit
plot_start_date = datetime(2022, 1, 1)
comparison_df = pd.concat([res.derived_outputs, cases_data])
comparison_df.columns = ['modelled', 'reported']
comparison_plot = comparison_df.plot(xlim=(plot_start_date,
 ↪  analysis_end_date))
```

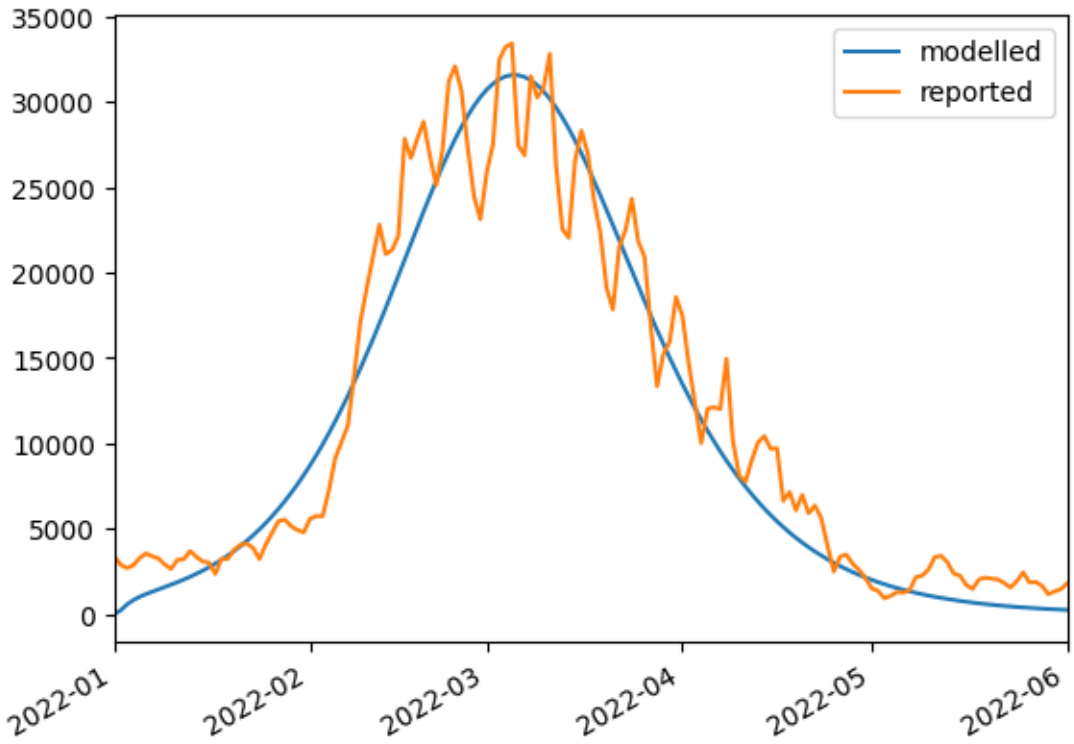


Figure 23: The calibration process involves using an optimisation algorithm to find the best-fit parameters for the epidemiological model. Running the model with these optimised parameters provides a prediction that closely matches the observed data. The comparison plot is a crucial tool for evaluating the model's performance, ensuring its predictions are reliable and accurate for public health planning and intervention strategies.

Perform calibration using parameter sampling

Performing calibration using parameter sampling provides a more comprehensive understanding of the model's parameters and their associated uncertainties. This approach enhances the model's ability to generate realistic predictions and account for the inherent variability in epidemiological processes.

- **Parameter Sampling**: Instead of finding a single "best" value for each parameter, this method explores a range of plausible parameter values based on prior knowledge and observed data. It generates a distribution of possible parameter values, reflecting the uncertainty inherent in the calibration process.

Common Techniques: * **Markov Chain Monte Carlo (MCMC)**: A popular method for sampling from the posterior distribution of parameters, which represents the updated beliefs about parameter values after considering the observed data.

Benefits of Parameter Sampling: * **Uncertainty Quantification**: Provides a measure of uncertainty associated with each parameter estimate, allowing for more robust and realistic predictions. * **Exploration of Parameter Space**: Explores a wider range of parameter values, potentially revealing multiple plausible scenarios that fit the data. * **Improved Model Robustness**: Leads to a more robust model that accounts for uncertainty in the input parameters.

```
idata = calibrate_model_with_sampling(bcm=bcm)
```

```
Multiprocess sampling (4 chains in 4 jobs)
DEMetropolisZ: [contact_rate, detection_prop]
```

```
Output()
```

```
Sampling 4 chains for 2_000 tune and 2_000 draw iterations (8_000 + 8_000 draws total) took 3
```

```
idata.posterior.to_dataframe()
```

chain	draw	contact_
0	0	0.39367
	1	0.39367
	2	0.39367
	3	0.39367
	4	0.39367
...	...	...

chain	draw	contact_
3	1995	0.38771
	1996	0.38771
	1997	0.38771
	1998	0.38771
	1999	0.36632

Figure 23: The detailed view of posterior samples provides insight into the distribution of parameter values obtained from the calibration process. The variability in the samples reflects the uncertainty in the parameter estimates, offering a more nuanced understanding of the parameter space. This detailed analysis enhances the accuracy and reliability of the epidemiological model, supporting informed public health decision-making and interventions.

```
# Arviz is a library for examining MCMC outputs and producing plots/summary
 ↪  statistics/wrangling data
# It supports a variety of frameworks (including PyMC, which we use above for
 ↪  producing our samples)
import arviz as az

az.plot_pair(idata);
```

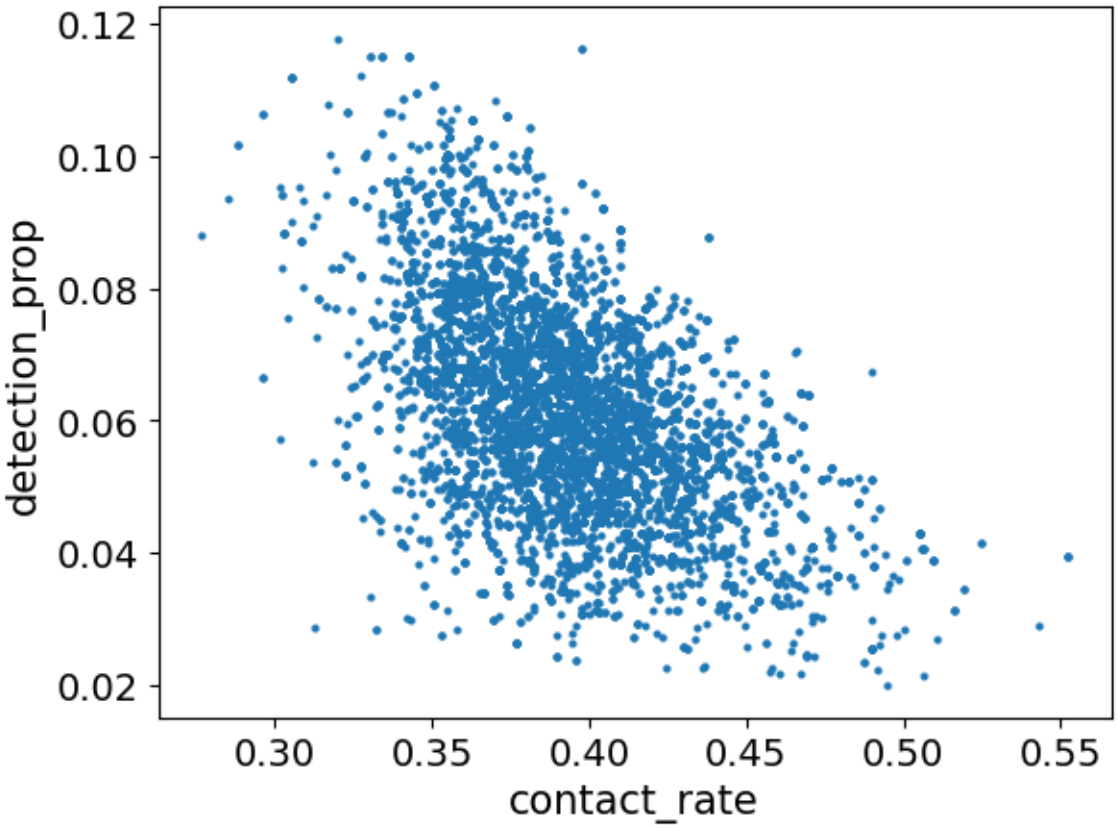


Figure 24: The scatter plot visualises the posterior distribution of contact_rate and detection_prop, providing insights into the parameter estimates and their variability. This visualization helps in understanding the potential correlation between parameters and the uncertainty associated with the estimates. The use of Arviz for plotting enhances the ability to analyze and interpret the results from the sampling process, supporting robust and informed calibration of the epidemiological model.

```
# Run the model for all parameter samples
# Estival includes a variety of tools for working with parameter sets
from estival.sampling import tools as esamptools
mres = esamptools.model_results_for_samples(idata, bcm)

# Note that we have run the model for all the samples here. In practice: we often select a subset of samples.
```

```
# Compare model output against data for a single selected sample.
sample_number = 220
```

```
plot_start_date = datetime(2021, 10, 1)
comparison_plot = pd.DataFrame({
    "modelled": mres.results['notifications', 0, sample_number],
    "reported": cases_data
}).plot(xlim=(plot_start_date, analysis_end_date))
```

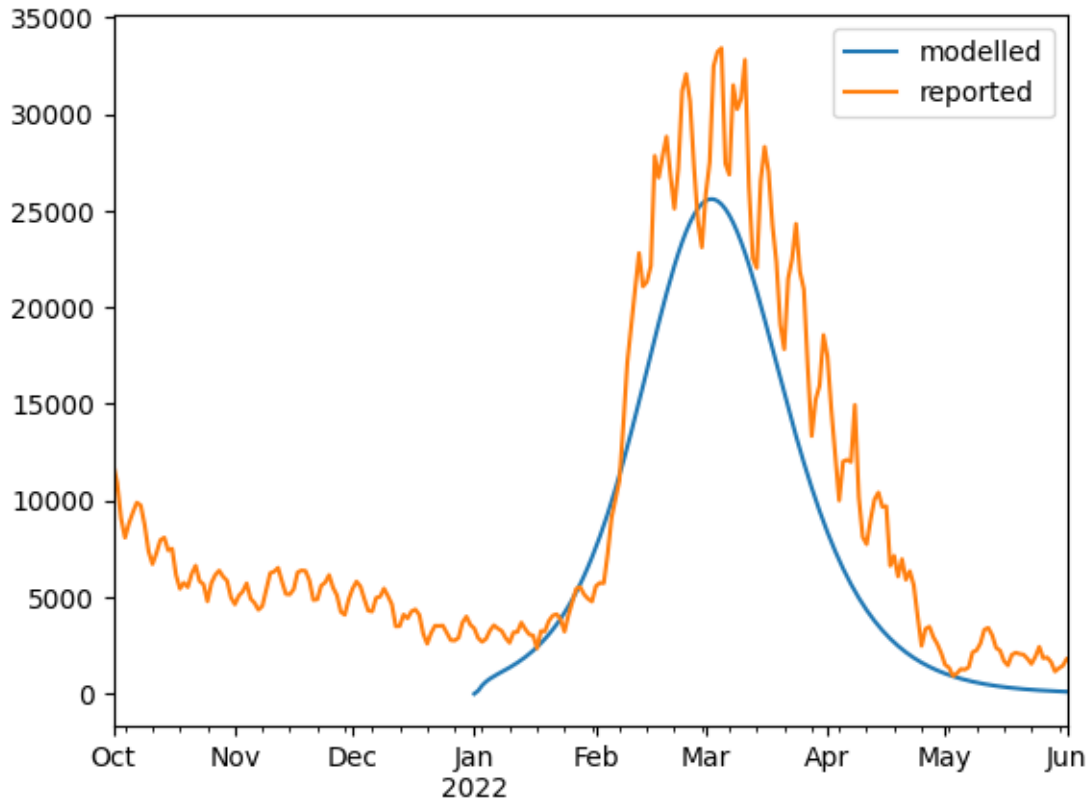


Figure 25: The process involves running the epidemiological model using multiple parameter samples from the posterior distribution and comparing the model output against observed data for a selected sample. The plot provides a visual comparison of modelled and reported cases, helping to evaluate the model's accuracy for specific parameter sets. This approach supports thorough analysis and validation of the model, enhancing its reliability for predicting disease spread and informing public health interventions.

This following code snippet visualizes the quantiles of model predictions for "notifications" alongside observed case data, temporarily switching the plotting backend for compatibility:

1. **pd.options.plotting.backend="matplotlib"**: Temporarily sets the plotting backend for pandas to "matplotlib." This is likely done to ensure compatibility with the esamp-

tools.quantiles_for_results function, which might be designed to work with matplotlib plots.

2. **esamptools.quantiles_for_results(mres.results,[0.025,0.25,0.5,0.75,0.975])["notifications"].pl**

- Calculates quantiles for the "notifications" output from the model results (mres.results) using the esamptools.quantiles_for_results function. The specified quantiles (0.025, 0.25, 0.5, 0.75, 0.975) represent the 2.5th percentile, 25th percentile, median, 75th percentile, and 97.5th percentile, respectively.
- Extracts the quantiles for "notifications" and generates a plot. This plot likely shows the range of predicted values for notifications with uncertainty bands.

3. **cases_data_select.plot(style='.',color='black')**: Plots the observed case data (cases_data_select) as black dots on the same plot. This allows for a direct visual comparison between the model's predicted notification quantiles and the actual observed cases.

In summary, this code generates a plot that visualizes the uncertainty in model predictions for notifications by showing quantiles and overlays the observed case data for comparison. The temporary switch to the matplotlib backend ensures compatibility with the quantile calculation function. This visualization helps assess the model's performance and the level of confidence in its predictions.

```
# Plot the uncertainty quantiles of the above sampled model results, against
 ↪  case data
esamptools.quantiles_for_results(mres.results,[0.025,0.25,0.5,0.75,0.975])["notifications"].p
cases_data_select.plot(style='.',color='black')
```

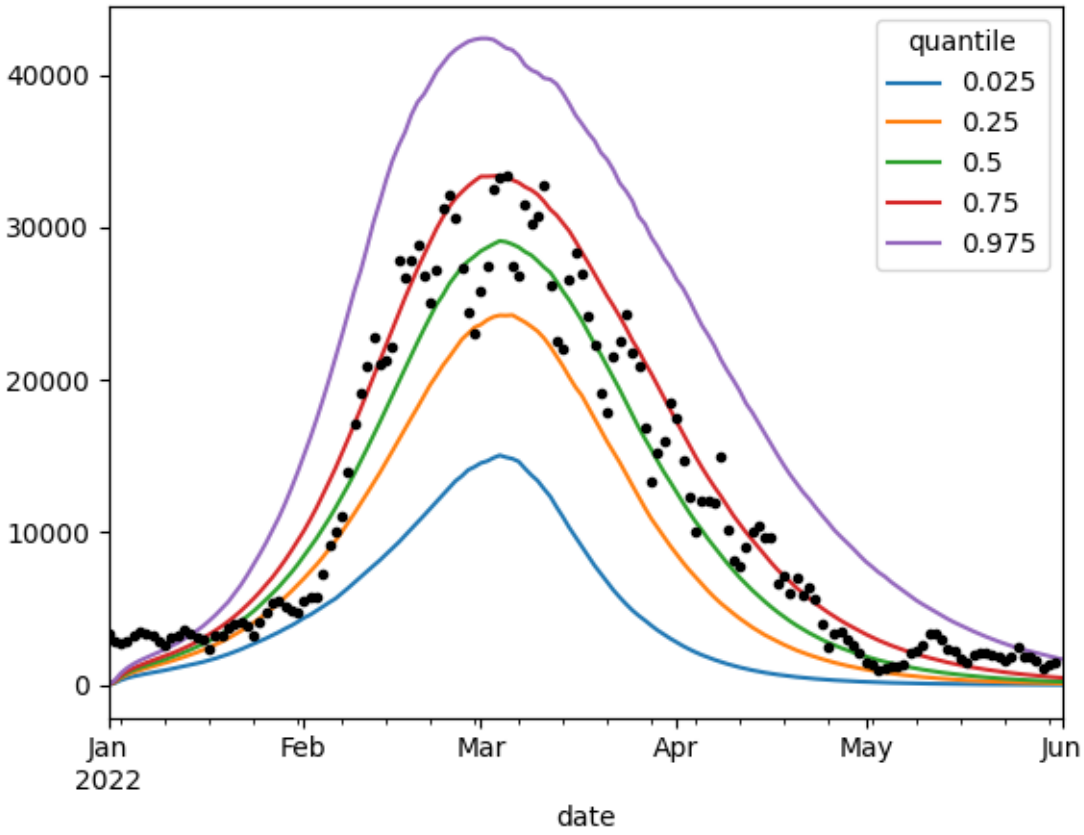


Figure 26: The plot compares the quantiles of the model predictions against the actual case data, providing a detailed view of the model's uncertainty and accuracy. The various quantile lines represent different levels of uncertainty, while the black dots show the observed case data. This visualization aids in assessing the model's predictive performance, highlighting the range of possible outcomes and the central tendency of the epidemic curve. Such analysis is crucial for robust public health decision-making and planning, ensuring that the model's predictions are reliable and informative.

Exploring more details about the parameter samples

By exploring more details about the parameter samples, you gain a deeper understanding of the model's behavior, its uncertainties, and its ability to capture the dynamics of the epidemiological system being studied.

- **Understanding Uncertainty**: Examining parameter samples helps quantify the uncertainty associated with each parameter estimate. This uncertainty reflects the range of plausible values for each parameter, given the observed data and prior knowledge.

- **Identifying Correlations**: Analyzing parameter samples can reveal correlations between different parameters. This information helps understand how changes in one parameter might affect others, providing insights into the model's dynamics.
- **Assessing Model Fit**: By comparing the distribution of parameter samples to prior assumptions, you can assess how well the model fits the observed data and whether any adjustments to the model structure or priors might be necessary.
- **Improving Predictions**: A deeper understanding of parameter samples can lead to more robust and reliable predictions, as it allows for incorporating uncertainty into future projections.

Common methods for exploring parameter samples:

- **Visualizations**: Histograms, scatter plots, and pair plots can be used to visualize the distribution of parameter samples and identify correlations.
- **Statistical Summaries**: Calculating summary statistics like mean, median, standard deviation, and quantiles provides a quantitative overview of the parameter samples.
- **Convergence Diagnostics**: Assessing the convergence of MCMC sampling methods ensures that the obtained samples accurately represent the posterior distribution of parameters.

```
# Get summary stats for the calibration
# These are useful to give a rough assessment of the quality of the outputs
az.summary(idata)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
contact_rate	0.392	0.032	0.338	0.457	0.001	0.001	989.0	1032.0	1.0
detection_prop	0.063	0.015	0.033	0.092	0.001	0.000	933.0	1311.0	1.0

This below code snippet generates a plot to visualize the trace of parameter samples obtained from a Bayesian inference process, using the ArviZ library:

1. **az.plot_trace(idata, ...)**: Calls the plot_trace function from the ArviZ library (az) to create a trace plot. The input idata is assumed to be an InferenceData object containing the parameter samples.
2. **figsize=(16,3.2*len(idata.posterior))**: Sets the figure size for the plot. The width is fixed at 16 inches, while the height is dynamically calculated based on the number of parameters in the posterior distribution (len(idata.posterior)), ensuring that each parameter gets adequate space in the plot.
3. **compact=False**: Disables the compact layout for the trace plot. This means that each parameter will be displayed in a separate subplot, providing a clearer view of individual parameter traces.

Reasoning: Trace plots are commonly used in Bayesian analysis to visualize the distribution and evolution of parameter samples generated by MCMC methods. They show the sampled values for each parameter over the course of the sampling process.

Key features of trace plots:

- **Distribution**: The distribution of the sampled values provides an indication of the posterior distribution of each parameter.
- **Convergence**: A well-mixed trace plot, where the samples appear to randomly explore the parameter space, suggests that the MCMC chains have converged to the target distribution.
- **Correlations**: Visual inspection of trace plots can reveal potential correlations between parameters.

By generating a trace plot with this code, you can visually assess the quality of the parameter samples, check for convergence, and gain insights into the posterior distribution of the model's parameters.

```
# Plot the traces (the values of the parameters at each sampling iteration)
az.plot_trace(idata, figsize=(16,3.2*len(idata.posterior)),compact=False);
```

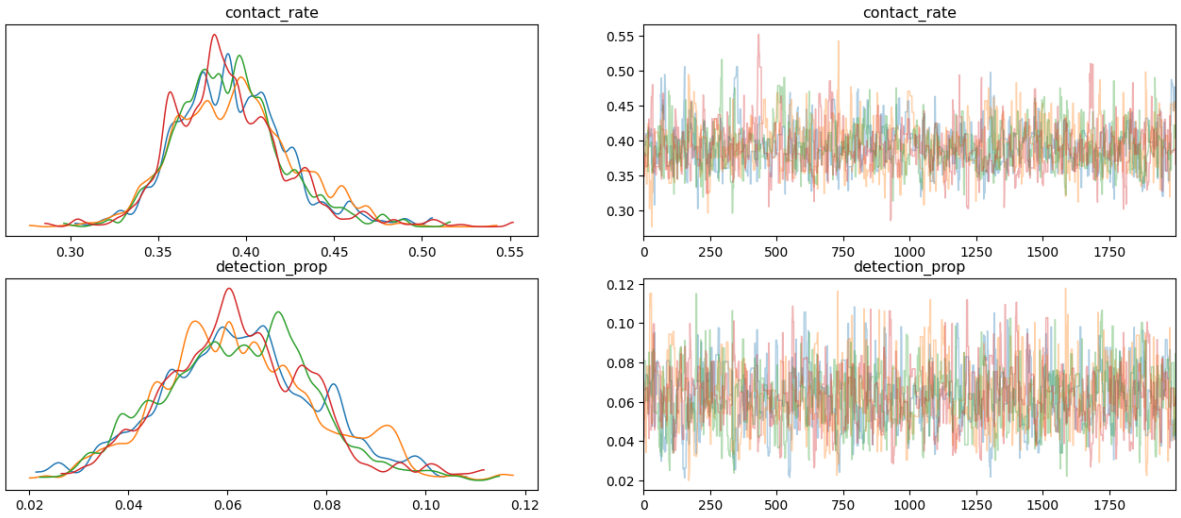


Figure 27: These visualisations are crucial for validating the results of the sampling process and understanding the uncertainty and variability in the parameter estimates, supporting robust and informed decision-making in epidemiological modeling.

This code snippet generates a plot to visualize the posterior distribution of parameters estimated from a Bayesian inference process:

1. **az.plot_posterior(idata)**: Calls the plot_posterior function from the ArviZ library (az) to create a posterior plot. The input idata is assumed to be an InferenceData object containing the parameter samples obtained from the Bayesian inference.

Reasoning: Posterior plots are essential in Bayesian analysis to summarize the inferred distribution of model parameters after considering both prior knowledge and observed data.

Key features of posterior plots:

- **Distribution**: The plot displays the estimated probability density function (PDF) or histogram for each parameter, representing the range of plausible values and their relative likelihoods.
- **Uncertainty**: The spread of the posterior distribution reflects the uncertainty associated with each parameter estimate.
- **Credible Intervals**: The plot often includes credible intervals (e.g., 94% HDI), which indicate the range within which the true parameter value is likely to fall with a certain probability.

By generating a posterior plot with this code, you can gain a visual understanding of the estimated parameter values, their uncertainties, and the overall shape of the posterior distribution. This information is crucial for interpreting the results of the Bayesian inference and drawing meaningful conclusions about the model and the system it represents.

```
# Plot the parameters' posterior distributions
az.plot_posterior(idata);
```

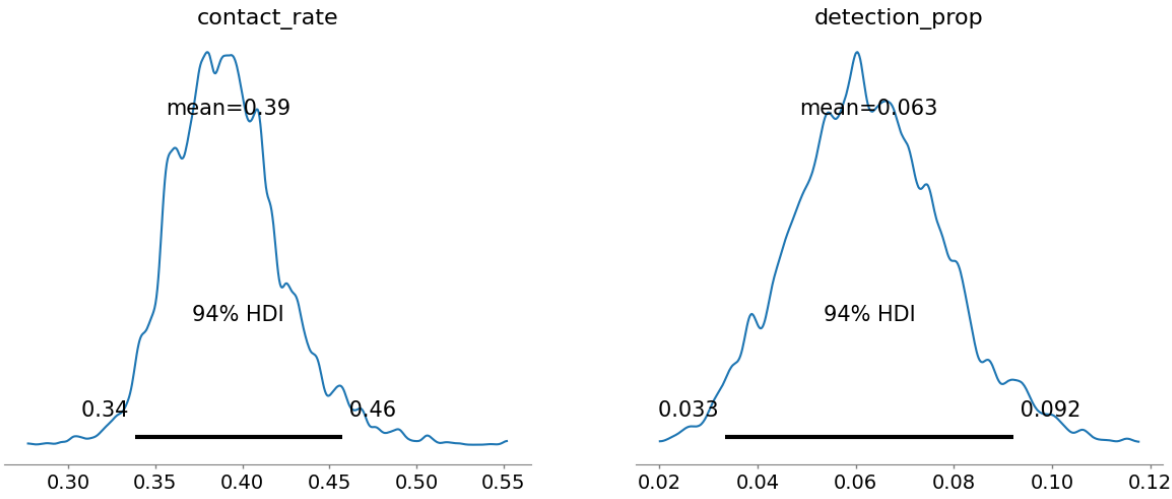


Figure 28: The posterior distribution plots for contact_rate and detection_prop illustrate the central estimates and the uncertainty associated with these parameters. The mean values and 94% HDIs provide comprehensive information about the parameter estimates, supporting

robust and informed decision-making. Understanding these distributions is crucial for evaluating the reliability and accuracy of the epidemiological model, ensuring it effectively captures the dynamics of disease transmission and detection.

Heterogeneous mixing

Heterogeneous mixing, in the context of infectious disease modeling, refers to the concept that individuals in a population do not interact randomly with each other.

Reasoning: Instead, their interactions are influenced by factors like age, social groups, location, and behavior.

Here's why heterogeneous mixing is important:

- **Realistic Disease Spread**: Captures the fact that people have different contact patterns, leading to more accurate simulations of how a disease spreads.
- **Targeted Interventions**: Allows for evaluating the effectiveness of interventions tailored to specific subgroups with different mixing patterns.
- **Understanding Transmission Dynamics**: Helps identify key drivers of disease transmission within different population groups.

Examples of heterogeneous mixing:

- **Age-structured models**: Consider different contact rates between children, adults, and the elderly.
- **Network models**: Represent individuals as nodes and their contacts as edges, capturing the complex structure of social interactions.
- **Spatial models**: Account for variations in contact patterns based on geographical location.

By incorporating heterogeneous mixing into infectious disease models, you can achieve a more nuanced and realistic representation of disease transmission dynamics, leading to better insights and more effective public health strategies.

Force of infection

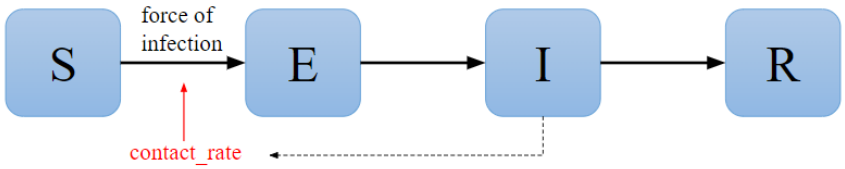


Figure 21: image.png

Figure 29: The force of infection, influenced by the contact rate, determines the rate at which susceptible individuals become exposed to the infection. Understanding the force of infection is crucial for accurately modeling and predicting the spread of infectious diseases, allowing for effective public health interventions and control measures.

Force of Infection

The force of infection represents the rate at which susceptible individuals contract the infection. It is a crucial component in the SEIR model, influencing the transition from the Susceptible (S) compartment to the Exposed (E) compartment.

Frequency-Dependent Transmission

In the context of frequency-dependent transmission, the force of infection ( ) is given by the formula:

$$\lambda = \beta \frac{I}{N}$$

Figure 22: image.png

Where:

- is the contact rate or transmission coefficient.
- I is the number of infectious individuals.
- N is the total population.

This formula indicates that the force of infection depends on the contact rate and the proportion of the population that is infectious.

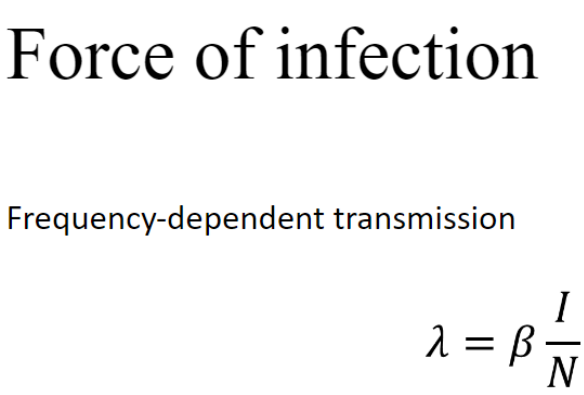


Figure 23: image.png

Mixing Matrix

The mixing matrix is used to represent the contact patterns between different subpopulations. It helps in understanding how individuals from different groups interact with each other. In the provided matrix:

- Rows represent the group from which an individual is from (north or south).
- Columns represent the group with which they are in contact (north or south).

Each cell in the matrix (a, b, c, d) represents the average number of contacts per time unit between individuals from the corresponding groups. For example:

- If an individual is from the north, they have a contacts with individuals from the north and b contacts with individuals from the south.

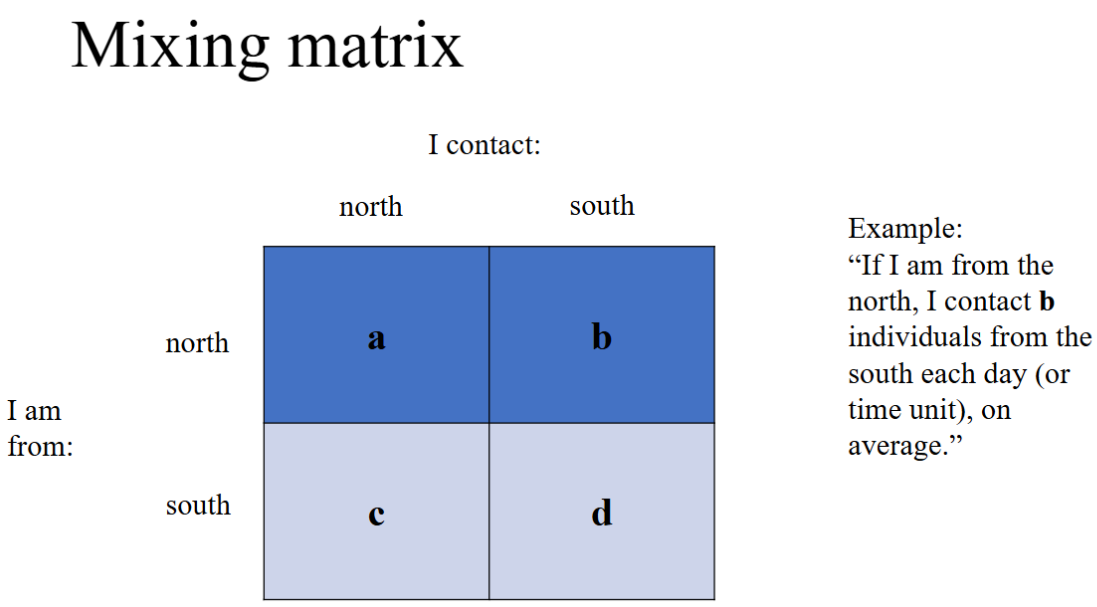


Figure 24: image.png

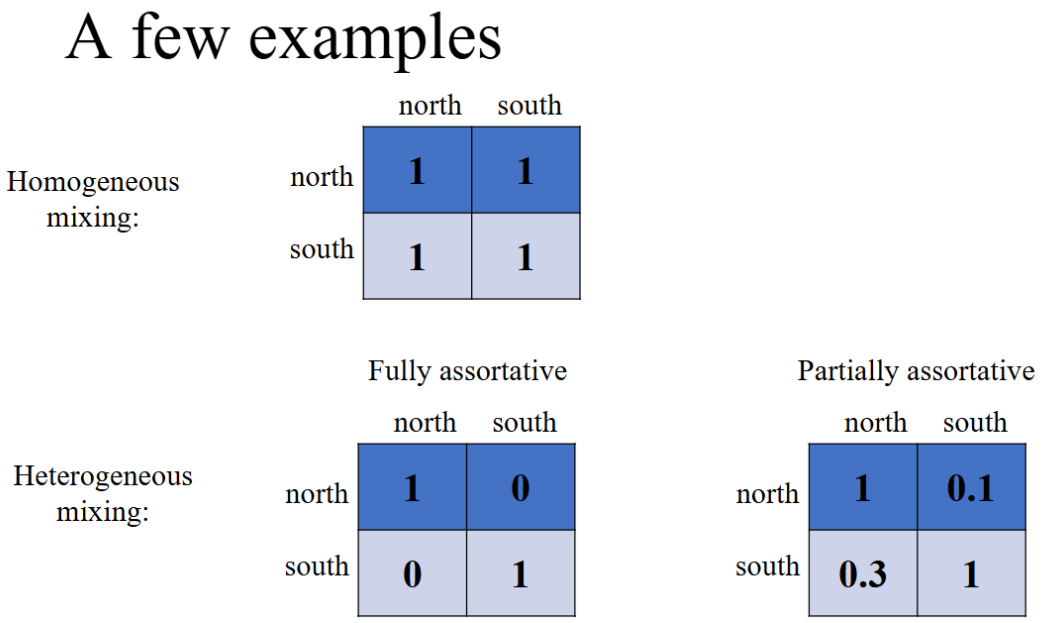


Figure 25: image.png

Examples of Mixing Patterns

The image provides different examples of mixing patterns:

1. Homogeneous Mixing:

- All individuals mix uniformly, regardless of their group.
- The mixing matrix values are all equal (e.g., 1).

2. Fully Assortative Mixing:

- Individuals mix only within their own group.
- The mixing matrix has 1s on the diagonal (contacts within the same group) and 0s off the diagonal (no contacts between different groups).

3. Partially Assortative Mixing:

- Individuals have a higher tendency to mix within their own group but also have some contacts with individuals from other groups.
- The mixing matrix has values on the diagonal greater than those off the diagonal, indicating more within-group contacts and fewer between-group contacts.

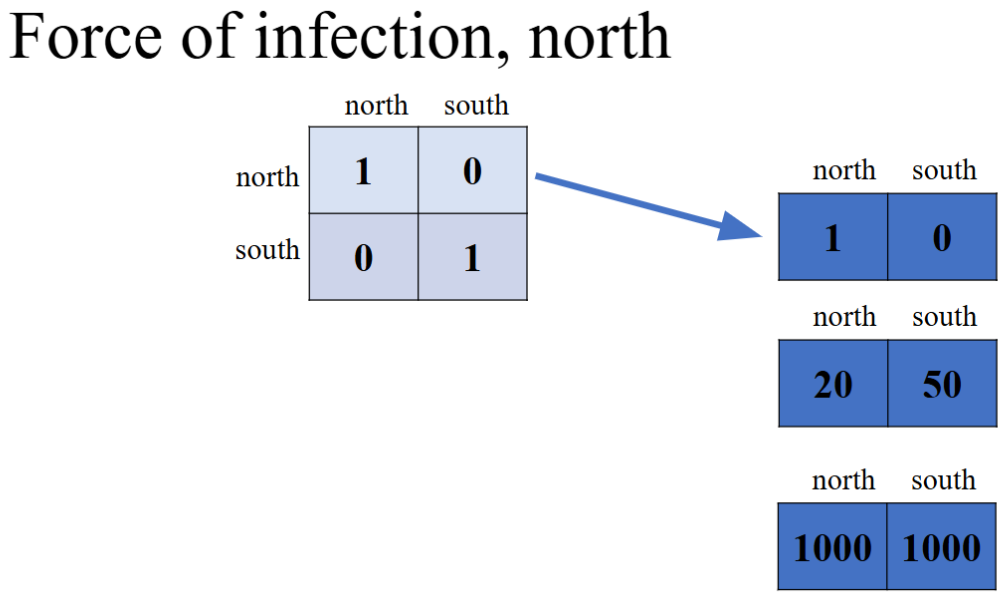


Figure 26: image.png

The above image shows a force of infection scenario, specifically focused on the “north” region. It presents a 2x2 matrix representing the force of infection, where:

- North to North: The value is 1, indicating a strong force of infection within the north region itself.
- North to South: The value is 0, indicating no infection spread from north to south.
- South to North: The value is 0, indicating no infection spread from south to north.
- South to South: The value is 1, indicating a strong force of infection within the south region itself.

The second matrix reflects the population distribution after infection:

- North: The population remains at 1, implying no migration from north to south.
- South: The population increases to 50, suggesting a significant migration from north to south due to the strong infection force within the south region.

The scenario highlights the impact of a strong infection force within a specific region on population distribution, showcasing a potential migration pattern towards the region with stronger infection.

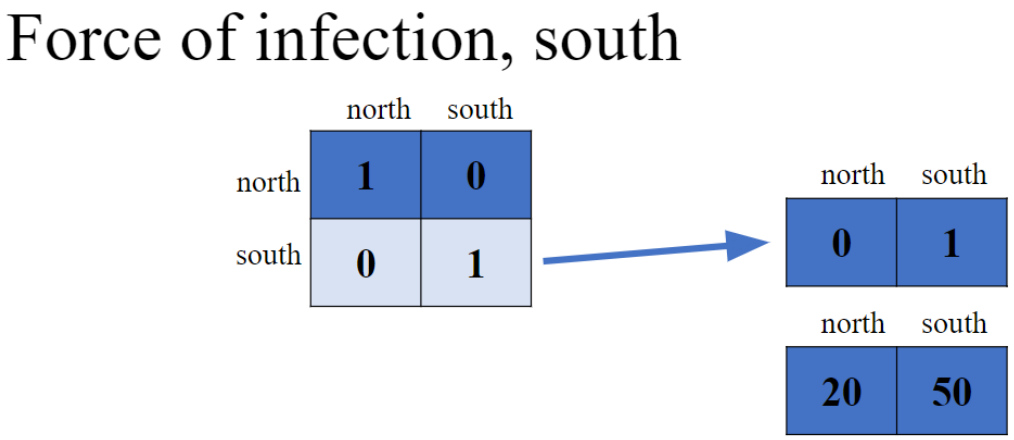


Figure 27: image.png

The resulting population distribution matrix shows the following:

- North: The population is 20, suggesting a decrease from the initial population.
- South: The population is 50, indicating an increase from the initial population.

Therefore, although there is no cross-region transmission of the infection, the strong force of infection within the south region has led to a migration pattern where individuals from the north have moved to the south. This suggests that, despite the absence of direct infection transmission between regions, the internal dynamics within the south region have resulted in a population shift.

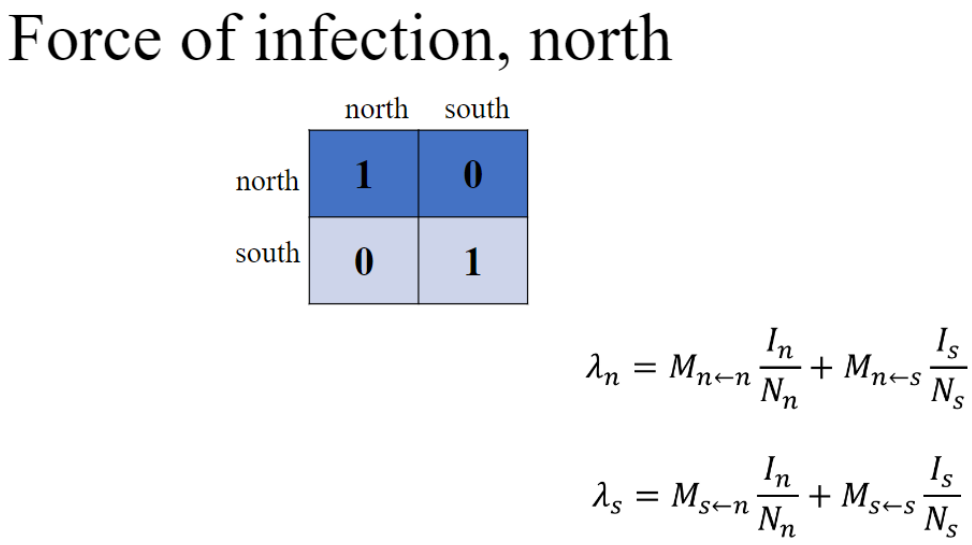


Figure 28: image.png

- M: Migration rate
- I: Number of infected individuals
- N: Total population

The force of infection in each region depends on the migration rate from that region (n→n, n→s, s→n, s→s), the number of infected individuals in both regions (In, Is), and the total population in both regions (Nn, Ns).

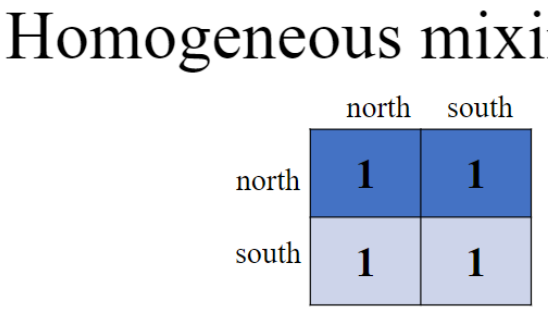


Figure 29: image.png

The above table displays a 2x2 matrix illustrating the force of infection, where:

- North to North: The value is 1, indicating an equal force of infection within the north region itself.
- North to South: The value is 1, indicating an equal force of infection from north to south.
- South to North: The value is 1, indicating an equal force of infection from south to north.
- South to South: The value is 1, indicating an equal force of infection within the south region itself.

There is no directional bias in the infection spread, meaning the infection spreads equally between and within the north and south regions. This concept is referred to as homogeneous mixing, implying that the infection is distributed uniformly across both regions.

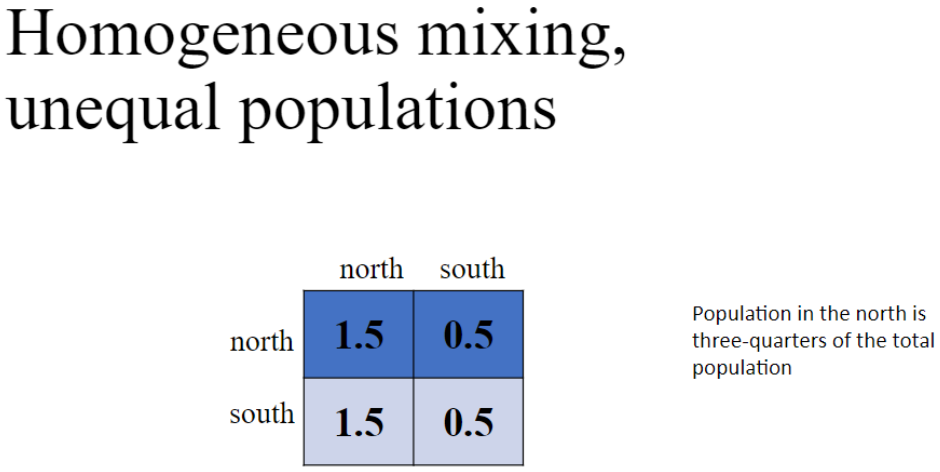


Figure 30: image.png

The above image illustrates a scenario of "homogeneous mixing" with "unequal populations" in two regions: north and south. It presents a 2x2 matrix representing the force of infection:

- North to North: The value is 1.5, indicating a stronger force of infection within the north region itself compared to other interactions.
- North to South: The value is 0.5, indicating a weaker force of infection from north to south.
- South to North: The value is 1.5, indicating a stronger force of infection from south to north.
- South to South: The value is 0.5, indicating a weaker force of infection within the south region itself.

This scenario highlights homogeneous mixing because the infection spreads equally in all directions despite the unequal population size. The population in the north is three-quarters of the total population, indicating a larger population in the north compared to the south. Despite this difference in population size, the infection spreads with equal force in all directions, emphasizing the concept of homogeneous mixing.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install summerepi2==1.3.5
```

```
import pandas as pd
from datetime import datetime
import numpy as np
from summer2 import CompartmentalModel, Stratification
from summer2.parameters import Parameter
```

Stratification with heterogeneous mixing

Stratification

This notebook applies a simple two-strata stratification to the basic model introduced in the SEIR model notebook. The two strata are arbitrarily referred to with compass bearings implying that they are intended to represent spatial stratification, but there is nothing else that implies that this population heterogeneity is particularly spatial.

Heterogeneous mixing

After creating the stratification object, a mixing matrix is then assigned. In this trivial example, the mixing matrix implies that mixing is completely assortative, with all contacts being made within an individual's stratum.

```
iso = 'MYS'
```

```
cases_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_cases.csv',
 ↪  index_col=0)[iso]
cases_data.index = pd.to_datetime(cases_data.index)
approx_pops = {
    'MYS': 33e6,
    'PHL': 114e6,
    'VNM': 97e6,
}
```

```
analysis_start_date = datetime(2021, 8, 1)
analysis_end_date = datetime(2022, 6, 1)
compartments = ['susceptible', 'exposed', 'infectious', 'recovered']
epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
    compartments,
```

```
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'infectious': 1.0})
incidence = epi_model.request_output_for_flow('incidence', 'progression',
 ↪  save_results=False)
epi_model.request_function_output('notifications', incidence *
 ↪  Parameter('detection_prop'));
```

```
# Stratification
spatial_strata = ['north', 'south']
spatial_strat = Stratification('regions', spatial_strata, compartments)
mixing_matrix = np.array(
    [
        [1.0, 0.0],
        [0.0, 1.0],
    ],
)
spatial_strat.set_mixing_matrix(mixing_matrix)
epi_model.stratify_with(spatial_strat)
for strat in spatial_strata:
    inc_string = f'incX{strat}'
    spatial_inc = epi_model.request_output_for_flow(inc_string,
 ↪  'progression', source_strata={'regions': strat}, save_results=False)
    epi_model.request_function_output(f'notifX{strat}', spatial_inc *
 ↪  Parameter('detection_prop'))
```

```
parameters = {
    'contact_rate': 0.4,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.07,
}
epi_model.run(parameters)
```

This code snippet generates an area plot to visualize the derived outputs of an epidemiological model, specifically focusing on notifications stratified by spatial regions, and overlays it with reported case data as markers:

1. **plot_start_date = datetime(2021, 10, 1)**: Sets the starting date for the plot's x-axis to October 1, 2021.
2. **plot = epi_model.get_derived_outputs_df()[[f'notifX{strat}' for strat in spatial_strata]].plot.area()**:

- Extracts the derived outputs from the epidemiological model (epi_model.get_derived_outputs_df()).
- Selects columns representing notifications for each spatial stratum using a list comprehension: [f'notifX{strat}' for strat in spatial_strata].
- Creates an area plot using the selected columns (plot.area()), where the area under each curve represents the cumulative number of notifications for each spatial region over time.

3. **plot.update_xaxes(range=(plot_start_date, analysis_end_date))**: Adjusts the x-axis to display the time range from plot_start_date to analysis_end_date.
4. **plot.add_trace(go.Scatter(x=cases_data.index, y=cases_data, name='data', mode='markers'))**:

- Adds a scatter plot trace to the existing area plot.
- Uses the go.Scatter function (likely from the Plotly library) to create a scatter plot with markers.
- Sets the x-values to the index of the cases_data dataframe (likely dates or time points).
- Sets the y-values to the actual reported case data (cases_data).
- Labels the scatter plot trace as 'data'.

In summary, this code generates an area plot showing the cumulative notifications for different spatial regions over time, based on the epidemiological model. It then overlays the reported case data as markers on top of the area plot, allowing for a direct comparison between the model's predictions and the actual observations. This visualization helps assess the model's ability to capture the dynamics of notifications and their distribution across different spatial locations.

```
plot_start_date = datetime(2021, 10, 1)
plot = epi_model.get_derived_outputs_df()[[f'notifX{strat}' for strat in
 ↪  spatial_strata]].plot.area(xlim=(plot_start_date, analysis_end_date))
cases_data.plot(style='.',color='black', xlim=(plot_start_date,
 ↪  analysis_end_date))
```

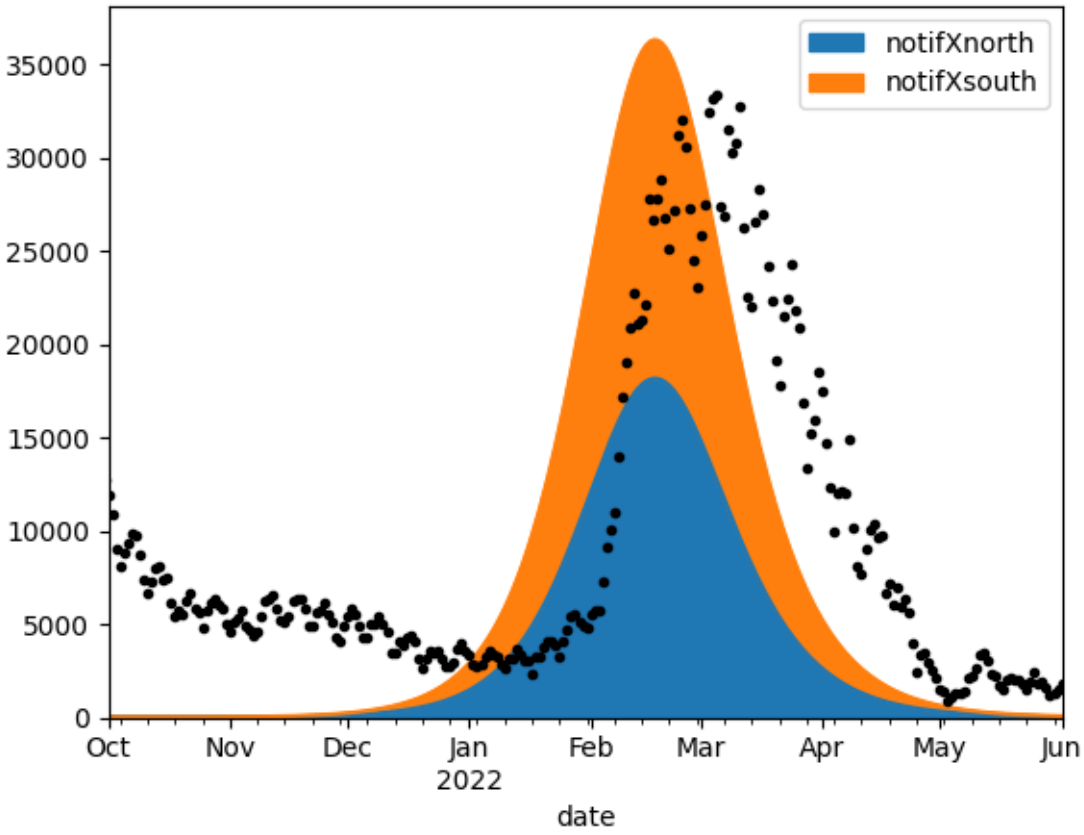


Figure 30: The figure compares model predictions for two regions (North and South) against actual reported data, providing a detailed view of the model's accuracy and regional differences. The shaded areas represent cumulative model predictions, while the green markers show actual cases. This comparison helps in assessing the model's performance and understanding regional variations in the epidemic's spread, supporting effective public health decision-making and intervention strategies.

Serial latent compartments

In infectious disease modeling, the concept of serial latent compartments refers to a way of representing the progression of an infection through multiple stages before an individual becomes infectious. This is often used to more accurately capture the dynamics of disease transmission, particularly for diseases where there is a significant delay between exposure and becoming infectious.

Latent Period: After an individual is exposed to an infectious agent, there is typically a period during which they are infected but not yet infectious. This is known as the latent

period.

Single vs. Serial Compartments:

- **Single Latent Compartment**: In a basic model, the latent period might be represented by a single compartment. Individuals move from the exposed (E) compartment directly to the infectious (I) compartment after a certain time.
- **Serial Latent Compartments**: To more accurately represent the distribution of the latent period, the latent stage is divided into multiple sub-stages or compartments. Individuals move through these compartments in sequence before reaching the infectious stage.

Mathematical Representation:

- **Basic SEIR Model**: In a simple SEIR (Susceptible, Exposed, Infectious, Recovered) model, an individual moves from S (susceptible) to E (exposed) to I (infectious) to R (recovered).
- **SEIR with Serial Latent Compartments**: Here, the exposed state (E) is split into several sub-compartments (E1, E2, …, En). This means an individual would progress through E1 to E2 to … En before becoming infectious (I).

Why Use Serial Latent Compartments?:

- **Realistic Latency Distribution**: It allows the model to better approximate the true distribution of the latent period, which often follows a gamma or Erlang distribution rather than an exponential distribution.
- **Improved Dynamics**: It can capture the fact that not all individuals become infectious at the same rate. This results in a more realistic depiction of how an infection spreads through a population.

Implementation:

- **Parameters**: The rate at which individuals move through the latent compartments is typically defined by a parameter that controls the transition between these stages.
- **Model Complexity**: Adding serial latent compartments increases the complexity of the model but improves its accuracy in predicting the spread of the infection.

Model specification

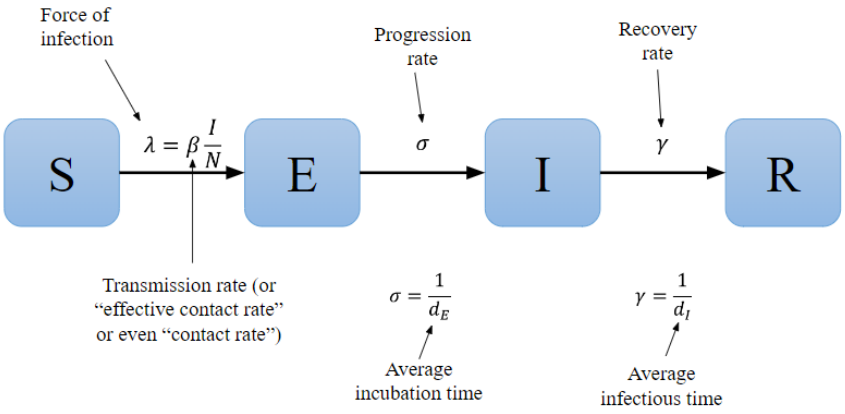


Figure 31: image.png

Figure 31: The force of infection, defined by the transmission rate and the proportion of infectious individuals, is a critical parameter in the model. The progression and recovery rates determine the speed at which individuals move through the exposed and infectious stages, respectively. Understanding these parameters is essential for accurately modeling disease dynamics and predicting the impact of interventions on disease spread.

Key parameters and rates:

- Force of Infection ( ): This is the rate at which susceptible individuals become exposed. It is given by $\lambda = \beta \frac{I}{N}$, where is the transmission rate (or effective contact rate), I is the number of infectious individuals, and N is the total population.
- Progression Rate ( ): The rate at which exposed individuals become infectious. It is the inverse of the average incubation time $(\sigma = \frac{1}{d_E})$.
- Recovery Rate ( ): The rate at which infectious individuals recover. It is the inverse of the average infectious time $(\gamma = \frac{1}{d_I})$.

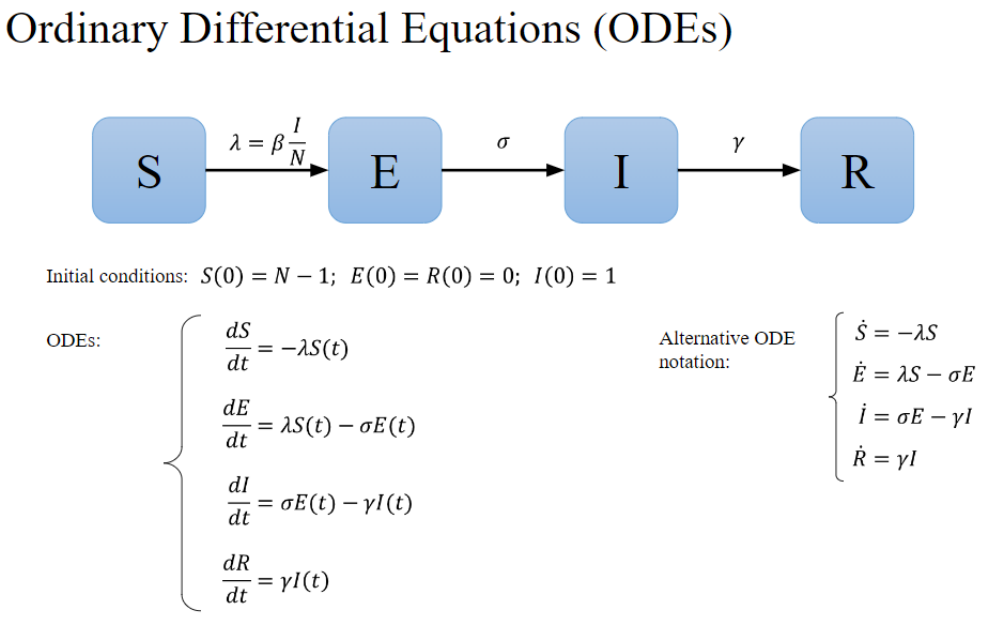


Figure 32: image.png

Figure 32: This figure presents the ordinary differential equations (ODEs) governing the SEIR (Susceptible-Exposed-Infectious-Recovered) model, which describes the dynamics of infectious disease spread through different compartments.

The ODEs for the SEIR model are as follows:

- $\frac{dS}{dt} = -\lambda S(t)$: The rate of change of susceptible individuals, which decreases as they become exposed.
- $\frac{dE}{dt} = \lambda S(t) - \sigma E(t)$: The rate of change of exposed individuals, which increases as susceptibles become exposed and decreases as exposed individuals become infectious.
- $\frac{dI}{dt} = \sigma E(t) - \gamma I(t)$: The rate of change of infectious individuals, which increases as exposed individuals become infectious and decreases as infectious individuals recover.
- $\frac{dR}{dt} = \gamma I(t)$: The rate of change of recovered individuals, which increases as infectious individuals recover.

Initial conditions:

- (0) = − 1 S(0)=N−1: Initial number of susceptible individuals, assuming one initial case. (0) = 0
- E(0)=0: Initial number of exposed individuals. (0) = 1
- I(0)=1: Initial number of infectious individuals. (0) = 0

- R(0)=0: Initial number of recovered individuals.

Alternative ODE notation:

- ˙ = − S ˙ =− S
- ˙ = − E ˙ = S− E
- ˙ = − I ˙ = E− I
- ˙ = R ˙ = I

This notation uses dots to indicate derivatives with respect to time.

These figures illustrate how the SEIR model can be mathematically formalized and solved using ODEs to simulate the dynamics of an infectious disease over time.

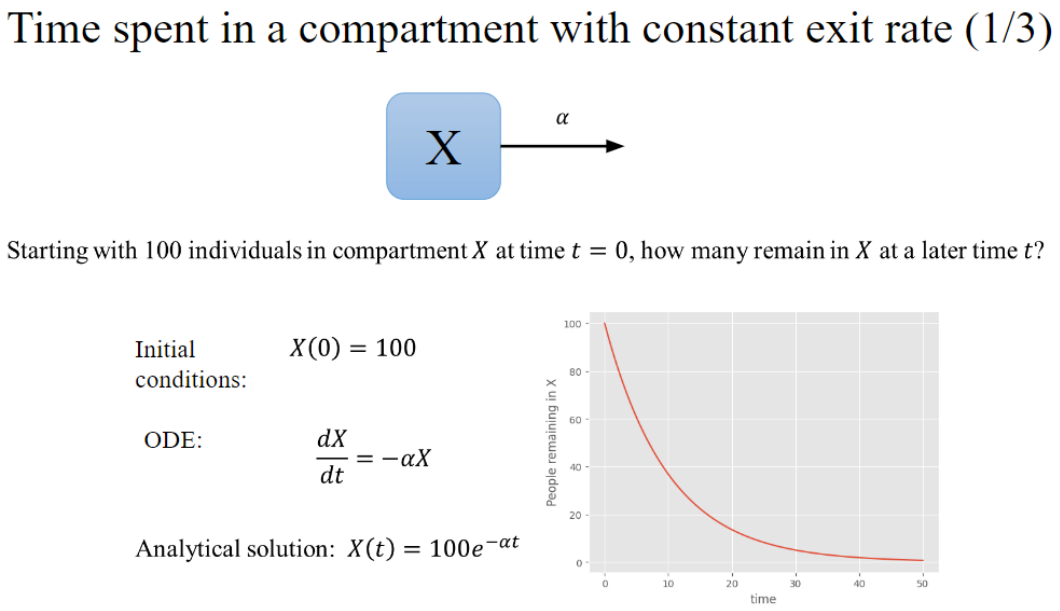


Figure 33: image.png

Figure 33: The figure demonstrates the concept of exponential decay in a compartmental model with a constant exit rate.

Key Components:

- Compartment X: Represents the number of individuals in the compartment at any given time.
- Exit Rate ( ): The rate at which individuals exit the compartment.

Initial Conditions:

- $(0) = 100$ X(0)=100: At time $= 0$ t=0, there are 100 individuals in compartment X.

Ordinary Differential Equation (ODE):

- $\frac{dX}{dt} = -\alpha X$: The rate of change of the number of individuals in the compartment is proportional to the current number of individuals in the compartment.

Analytical Solution:

- $X(t) = 100e^{-\alpha t}$ The number of individuals remaining in the compartment at time t follows an exponential decay, as shown in the graph.

The graph illustrates how the number of individuals in compartment X decreases over time according to the exponential decay function.

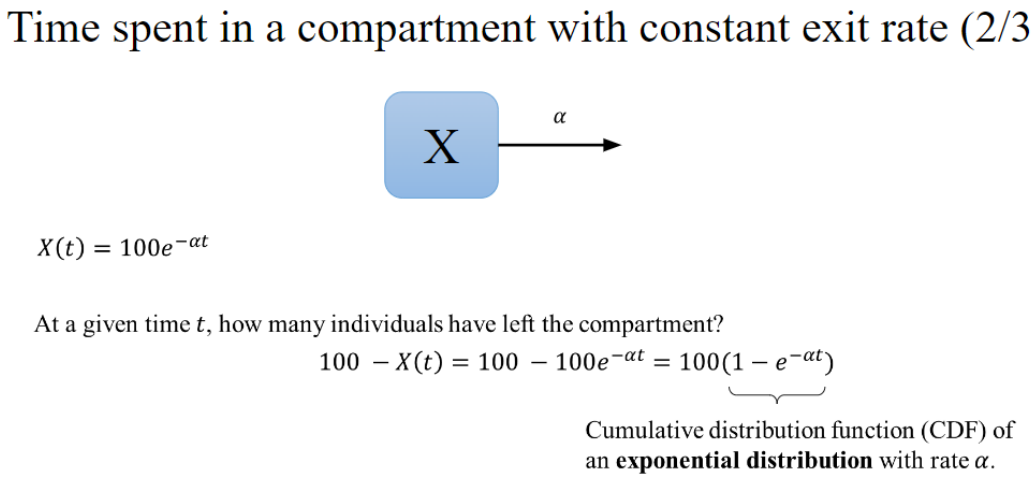


Figure 34: image.png

Figure 34: This figure further elaborates on the dynamics of individuals in a compartment with a constant exit rate , focusing on the number of individuals who have left the compartment over time.

Equation for Number of Individuals Remaining:

- $X(t) = 100e^{-\alpha t}$ The number of individuals remaining in the compartment at time .

Equation for Number of Individuals Who Have Left:

$$100 - X(t) = 100 - 100e^{-\alpha t} = 100(1 - e^{-\alpha t})$$

Figure 35: image.png

-

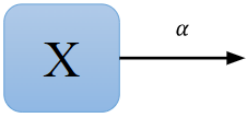
This formula represents the cumulative number of individuals who have left the compartment by time .

Key Point:

- The expression $100(1 - e^{-\alpha t})$ is the cumulative distribution function (CDF) of an exponential distribution with rate .

The cumulative distribution function (CDF) of an exponential distribution describes the probability that an event has occurred by a certain time , which in this context is the proportion of individuals who have exited the compartment by time .

Time spent in a compartment with constant exit rate (3/3)



Take-home messages:

- The time spent in a compartment (sojourn time) with constant exit rate α follows an **exponential distribution** with rate α.
- This means that even with a fixed parameter value, the model is capturing variation for the sojourn time.
- The mean of an exponential distribution with rate α is $\frac{1}{\alpha}$. This is why a compartment's exit rate is the reciprocal of the average duration spent in this compartment.

Figure 36: image.png

Can you think of any issues with the exponential distribution?

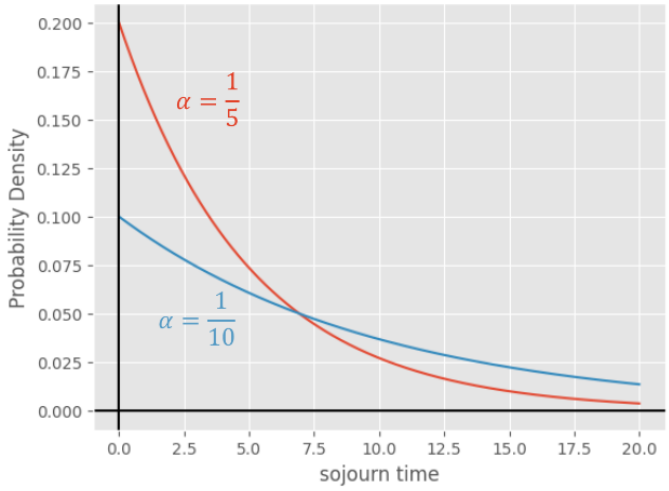


Figure 37: image.png

Figure 35: This figure presents a probability density function (PDF) for an exponential distribution with two different rates ($\alpha = \frac{1}{5}$ and $\alpha = \frac{1}{10}$) and poses the question about potential issues with using the exponential distribution to model sojourn time in a compartment.

The graph shows the probability density function (PDF) for two different exponential distributions with rates $\alpha = \frac{1}{5}$ and $\alpha = \frac{1}{10}$.

Key Points:

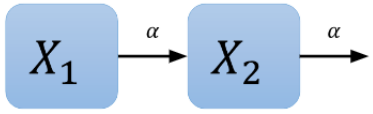
- The exponential distribution assumes a constant rate of exit, which may not accurately represent real-world scenarios where the rate might vary over time.
- The graph highlights that as the rate parameter changes, the shape of the distribution changes, indicating different decay rates for the probability density over time.

Solution: Combining multiple compartments to achieve different statistical distributions

For example, the sum of k exponential random variables with rate α is a Gamma random variable with parameters $(shape = k, rate = \alpha)$.

$$X_1, X_2, \ldots, X_k \sim Exp(\alpha) \quad \Longrightarrow \quad Y = X_1 + X_2 + \cdots + X_k \sim Gamma(k, \alpha).$$

The following structure models a sojourn time that is Gamma-distributed with parameters $(shape = 2, rate = \alpha)$



Note that a Gamma distribution is also called an "Erlang distribution" when the shape parameter is an integer.

Figure 38: image.png

- By structuring the model with multiple compartments, each with an exponential exit rate , the overall time spent in these compartments can be modeled with a Gamma distribution.
- This approach allows for more flexible modeling of the time distribution, accommodating scenarios where the rate of exit changes over time.

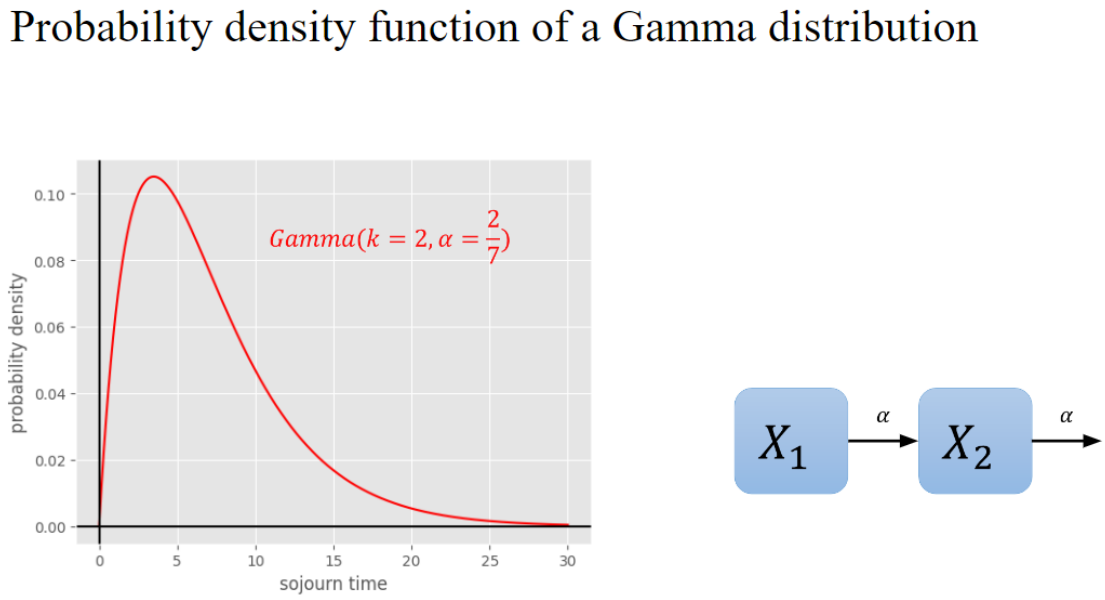


Figure 39: image.png

Figure 36: The figure shows the probability density function (PDF) of a Gamma distribution with parameters $k = 2$ and $\alpha = \frac{2}{7}$.

Key Points:

- The Gamma distribution, with its shape and rate parameters, provides a more flexible model for the time individuals spend in a compartment.
- The PDF graph illustrates how the Gamma distribution can capture different time distributions, unlike the single-parameter exponential distribution. Compartment Model:
- X_1 and X_2 compartments, each with exit rate , combine to form a Gamma-distributed sojourn time.

This method highlights how combining multiple exponential compartments can address the limitations of using a single exponential distribution, providing a more accurate representation of the time dynamics in epidemiological models.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install summerepi2==1.3.5
```

```
import pandas as pd
from datetime import datetime
from summer2 import CompartmentalModel
from summer2.parameters import Parameter
```

Epidemiological explanation

Here we take a similar epidemiological model to that presented in the seir_model.ipynb notebook and replicate the latent compartments in series.

Flow adjustments

In order to interpose additional latent or exposed compartments between the susceptible and infectious compartments, we need to make four structural adjustments to the model, as follows: - We remove the previous process of infection transfering newly infected persons to the single exposed compartment - The infection process needs to transfer newly infected person to the first of the sequential exposed compartments - We add transitions between each of the sequential latent compartments - We add a transition from the last of the latent compartments to the infectious compartment

Transition rates and parameters

We can retain the same parameters as in for the base SEIR model. However, this will result in much slower rates of arrival in the infectious compartment following infection if we retain the same rate parameter for each of the transfer processes. This is because infected persons will need to transition between each of the sequential compartments, and so only arrive in the infectious compartment once they complete these transition processes.

We can offset this by multiplying the rate of each transition by the number of transitions that must be made. This will result in the same mean time from infection to onset of infectiousness, such that we can still think of the parameter as the average incubation period. However, this will have implications for model dynamics, because the distribution of arrivals into the infectious compartment will differ from that under the base SEIR model. Specifically, the base case implies an exponential distribution for this time, whereas this model configuration implies an Erlang-distributed arrival time with shape parameter equal to the number of sequential compartments.

```
iso = 'MYS'
```

```
cases_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_cases.csv',
 ↪  index_col=0)[iso]
cases_data.index = pd.to_datetime(cases_data.index)
approx_pops = {
    'MYS': 33e6,
    'PHL': 114e6,
    'VNM': 97e6,
}
```

```
n_latent_comps = 3
latent_comps = [f'exposed_{i_comp}' for i_comp in range(n_latent_comps)]
compartments = ['susceptible', 'infectious', 'recovered'] + latent_comps
```

```
analysis_start_date = datetime(2021, 8, 1)
analysis_end_date = datetime(2022, 6, 1)
compartments = ['susceptible', 'infectious', 'recovered'] + latent_comps
epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
    compartments,
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
```

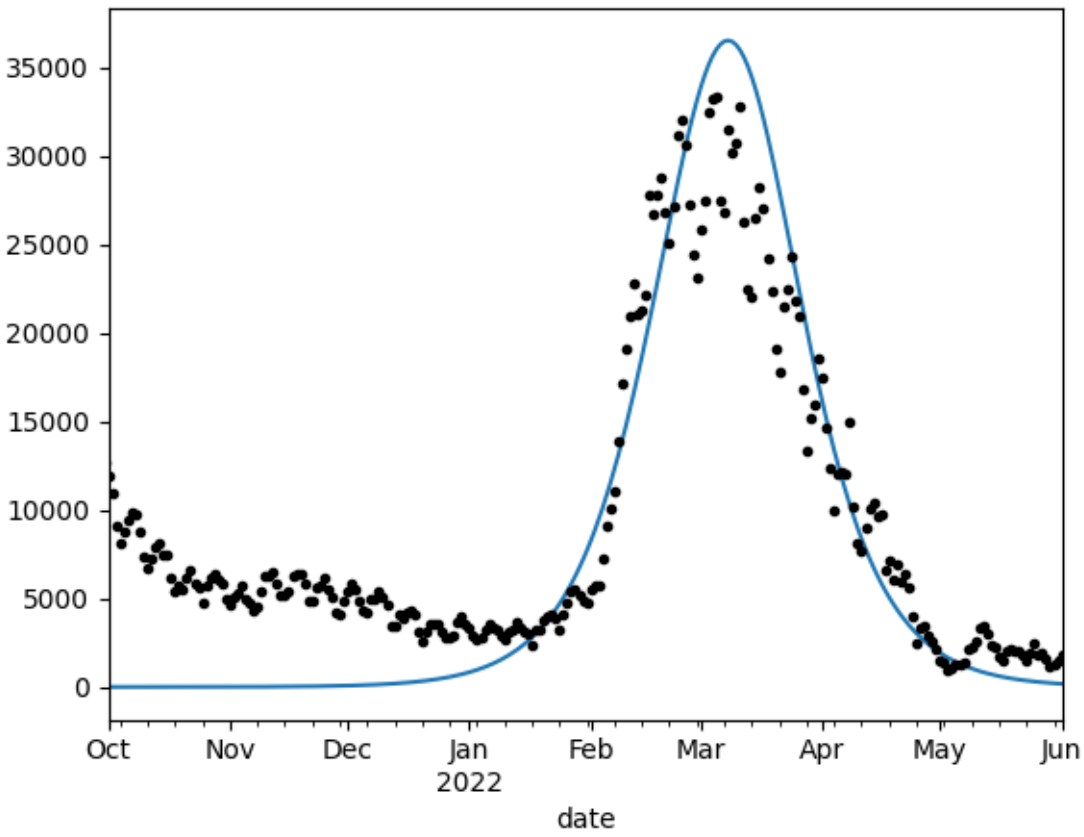
```
# Transition flows adjusted relative to the base SEIR model
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', latent_comps[0])
for i_comp in range(n_latent_comps - 1):
    source_comp = f'exposed_{i_comp}'
    dest_comp = f'exposed_{i_comp + 1}'
    transfer_rate = float(n_latent_comps) / Parameter('incubation_period')
    epi_model.add_transition_flow(f'exposed_transfer_{i_comp}',
 ↪  float(n_latent_comps) / Parameter('incubation_period'), source_comp,
 ↪  dest_comp)
epi_model.add_transition_flow('progression', float(n_latent_comps) /
 ↪  Parameter('incubation_period'), latent_comps[-1], 'infectious')
```

```
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'exposed_0': 1.0})
```

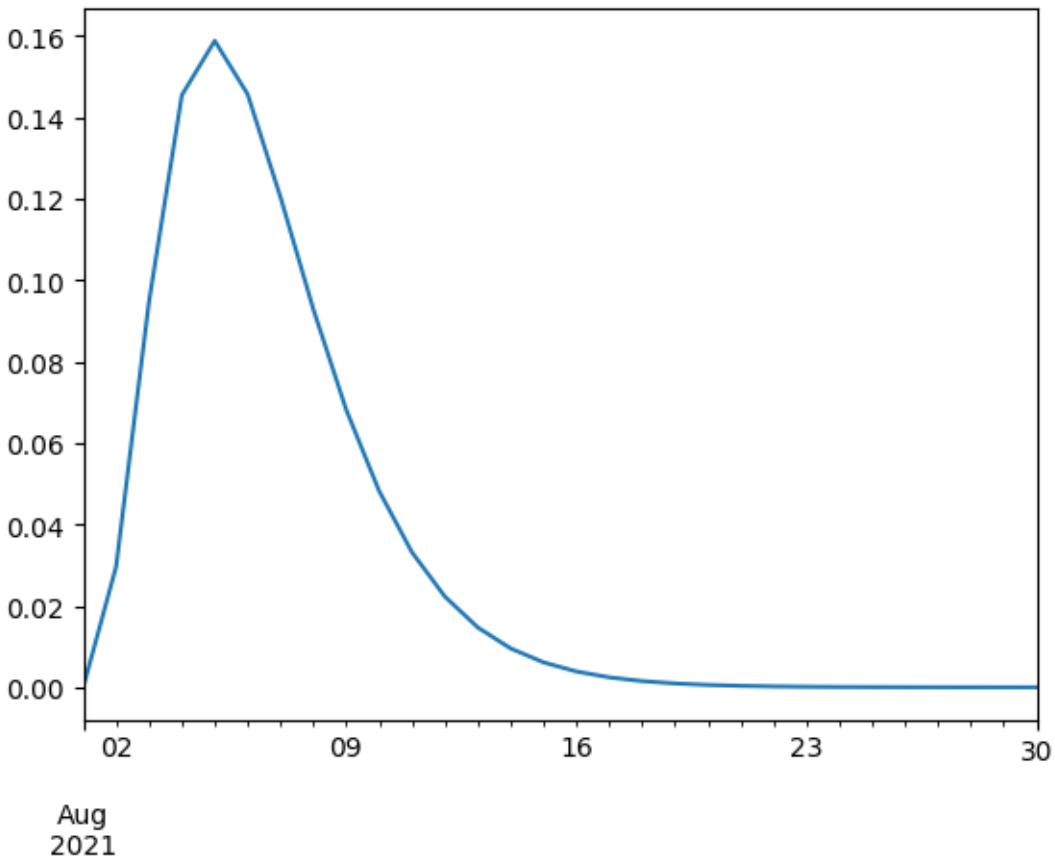
```
incidence = epi_model.request_output_for_flow('incidence', 'progression')
epi_model.request_function_output('notifications', incidence *
 ↪  Parameter('detection_prop'));
```

```
parameters = {
    'contact_rate': 0.4,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.07,
}
epi_model.run(parameters)
```

```
plot_start_date = datetime(2021, 10, 1)
comparison_plot =
 ↪  epi_model.get_derived_outputs_df()["notifications"][plot_start_date:analysis_end_date].pl
cases_data.plot(style='.',color='black',xlim=(plot_start_date,
 ↪  analysis_end_date))
```




```
epi_model.run(parameters | {"contact_rate": 0.0})
epi_model.get_derived_outputs_df()["incidence"].iloc[:30].plot()
```


0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
02
09
16
23
30
Aug
2021

Tracking model outputs

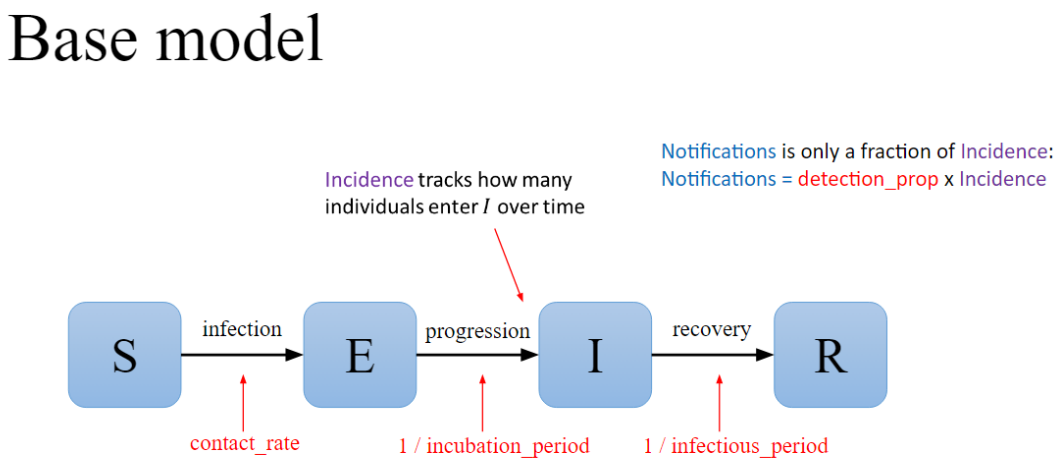


Figure 40: image.png

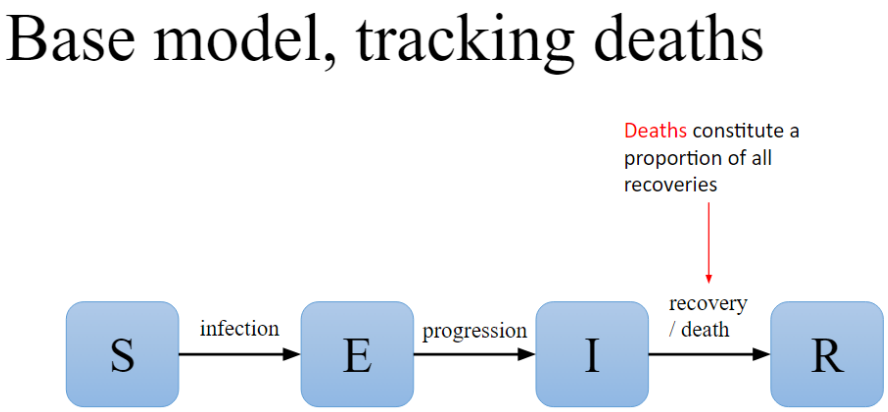


Figure 41: image.png

Figure 37: The figure extends the base SEIR model to track deaths. In this model:

- Individuals still move through the stages from susceptible (S) to exposed (E) to infectious (I) to recovered (R).
- However, the recovery stage (R) now also includes the possibility of death. Deaths are tracked as a proportion of the total recoveries.

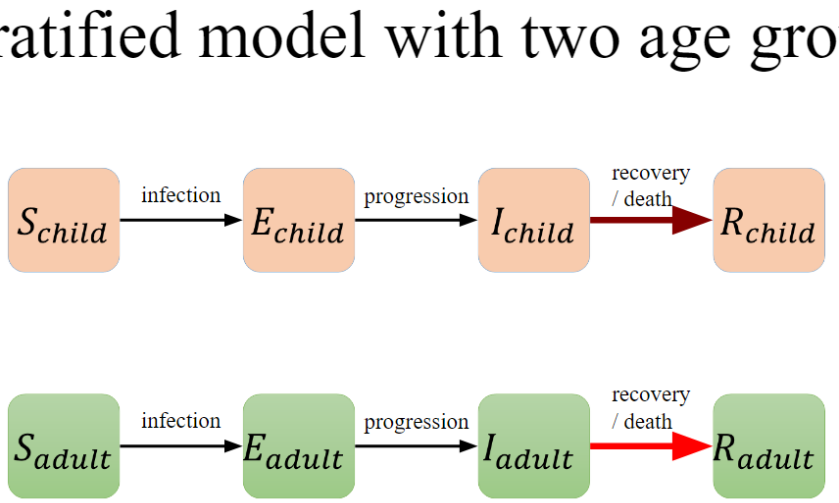


Figure 42: image.png

Figure 38: The figure presents a stratified SEIR model that distinguishes between two age groups: children and adults. This model involves:

- Separate compartments for each age group, including susceptible children (S_child) and adults (S_adult), exposed children (E_child) and adults (E_adult), infectious children (I_child) and adults (I_adult), and recovered (or dead) children (R_child) and adults (R_adult).

Each age group follows the SEIR progression with infection, progression from exposed to infectious, and then recovery or death. This stratification allows for different rates of infection, progression, recovery, and death for children and adults, which can be crucial for accurately modeling and understanding the dynamics of the disease across different segments of the population.

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
# !pip install summerepi2==1.3.5
```

```
import pandas as pd
from datetime import datetime
```

```
from summer2 import CompartmentalModel, Stratification
from summer2.parameters import Parameter, DerivedOutput
```

Epidemiological explanation

Here we take a similar epidemiological model to that presented in the `seir_model.ipynb` notebook and apply a simple stratification to it.

Age stratification

The stratification itself is named 'age', but does not incorporate demographic processes (because we only use the standard 'Stratification' object). The stratification has no epidemiological effect on the model behaviour, execept that it permits us to generate some age group-specific outputs (and in the second model apply some different flow rates).

Tracking death rates

The purpose of this stratification is to allow calculation of death rates, with the infection fatality rate (IFR) being allowed to differ according by age group. In the first model, we track recoveries and multiply the rate of recovery for each stratum by an age-specific IFR to calculate the total rate of deaths for the population. This means that we have a "closed population" and so just track deaths as a derived process linked to incidence.

Parameters

Some additional parameters are needed here, including the distribution of population by age group. None of the parameters are intended to be exact, but are presented to allow the user to adjust them to desired (and more realistic) ranges.

```
iso = 'MYS'
```

```
deaths_data =
 ↪  pd.read_csv('https://github.com/monash-emu/wpro_working/raw/main/data/new_deaths.csv',
 ↪  index_col=0)[iso]
deaths_data.index = pd.to_datetime(deaths_data.index)
approx_pops = {
    'MYS': 33e6,
    'PHL': 114e6,
```

```
    'VNM': 97e6,
}
analysis_start_date = datetime(2021, 8, 1)
analysis_end_date = datetime(2022, 6, 1)
```

```
def get_epi_model():
    epi_model = CompartmentalModel(
        [analysis_start_date, analysis_end_date],
        ['susceptible', 'exposed', 'infectious', 'recovered'],
        ['infectious'],
        ref_date=datetime(2019, 12, 31),
    )
    epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
    epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
    epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
    epi_model.set_initial_population({'susceptible': approx_pops[iso],
 ↪  'infectious': 1.0})
    return epi_model

age_strata = ['0', '15', '60', '75']
age_pop_split = {'0': 0.2, '15': 0.5, '60': 0.2, '75': 0.1}

def get_age_strat(comp_names):
    age_strat = Stratification('age', age_strata, comp_names)
    age_strat.set_population_split(age_pop_split)
    return age_strat

plot_start_date = datetime(2021, 12, 1)

def plot_age_deaths(model):
    fig = model.get_derived_outputs_df().plot.area(xlim=(plot_start_date,
 ↪  analysis_end_date))
    deaths_data.plot(style='.',color='black',xlim=(plot_start_date,
 ↪  analysis_end_date))
    return fig
```

```
# Get model and stratify by age
epi_model_1 = get_epi_model()
```

```
age_strat = get_age_strat(epi_model_1._original_compartment_names)
epi_model_1.stratify_with(age_strat)

# Calculate deaths as a proportion of the recovery flow
for age in age_strata:
    age_recoveries = epi_model_1.request_output_for_flow(f'rec_{age}',
 ↪  'recovery', source_strata={'age': age}, save_results=False)
    epi_model_1.request_function_output(f'deaths_{age}', age_recoveries *
 ↪  Parameter(f'ifr_{age}'))
```

```
# Specify parameters, run and plot outputs
parameters = {
    'contact_rate': 0.4,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'ifr_0': 0.0,
    'ifr_15': 0.0,
    'ifr_60': 0.0002,
    'ifr_75': 0.002,
}
epi_model_1.run(parameters)
plot_age_deaths(epi_model_1)
```

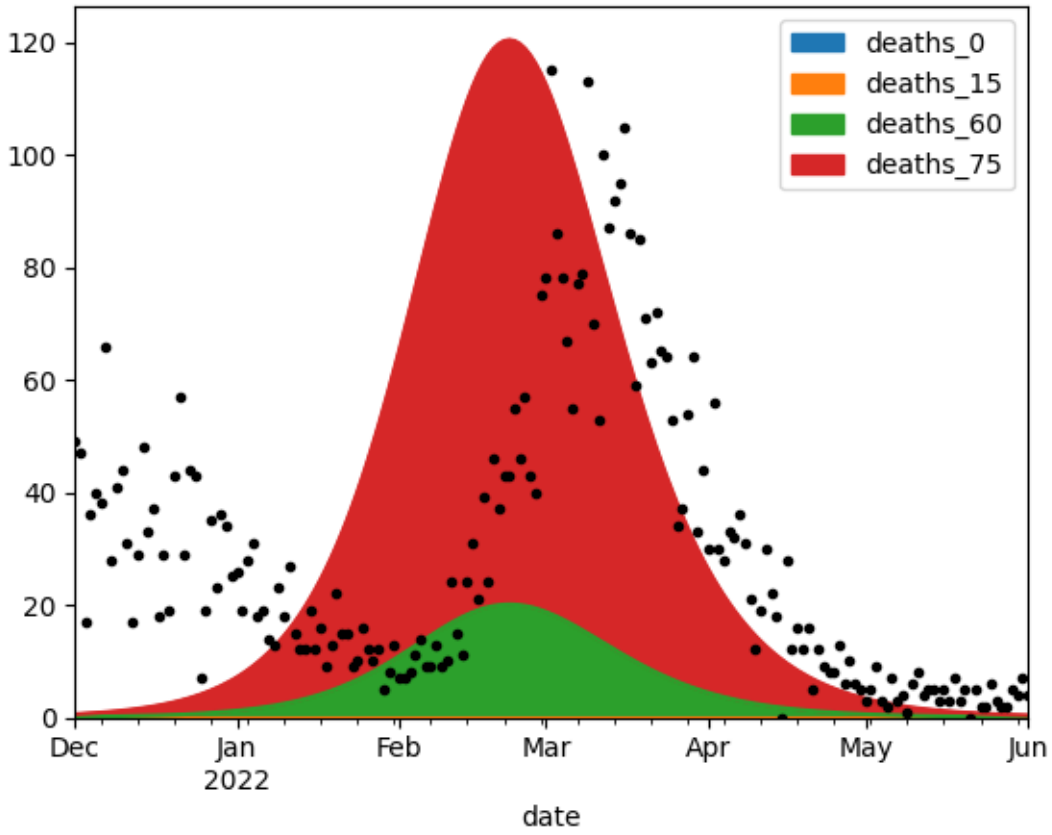


Figure 39: This figure shows the output of an age-stratified epidemiological model predicting the number of deaths across different age groups over time. The model parameters are specified, and the results are plotted and compared to reported deaths.

The age-stratified model uses specified parameters to predict the number of deaths across different age groups over time.

The plotted results compare the model's predictions with actual reported deaths, highlighting the model's performance and the impact of age-specific fatality rates. This analysis is crucial for understanding the differential impact of infectious diseases on various age groups and for planning targeted public health interventions.

Predicted vs. Reported Deaths:

- The model predictions (shaded areas) can be compared to the reported deaths (orange line) to evaluate the model's accuracy.
- The alignment or divergence between the predicted and reported deaths provides insight into the model's performance.

Age Stratification:

- Different age groups show varying levels of predicted deaths, reflecting the age-specific infection fatality rates (IFRs).
- Higher age groups (60 and 75) have higher predicted deaths, which is consistent with higher IFRs for older individuals.

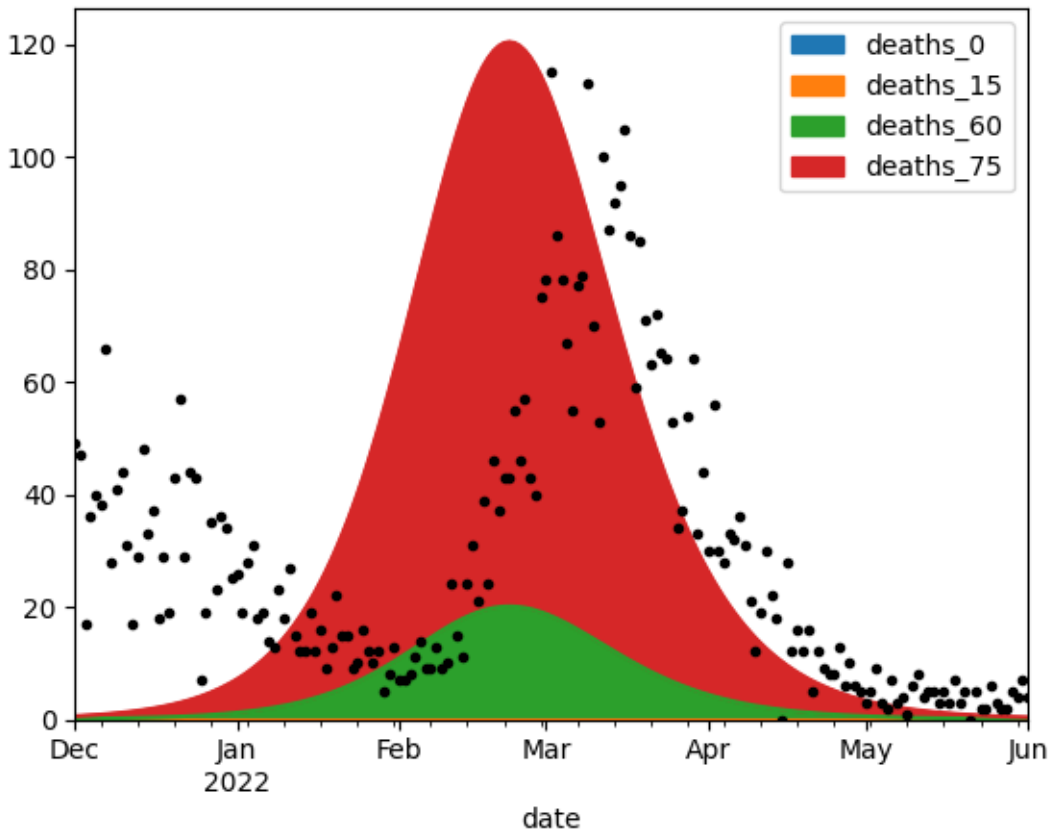
Using an explicit death flow

In this alternative version of the model, we will apply a death flow using `summer2`'s death flow method, which means that people actually leave the model as they die (and so the model is no longer "closed"). We'll run it with the same parameters and should expect similar (although not quite identical) dynamics.

```
# Get model and add an explicit deaths flow
epi_model_2 = get_epi_model()
epi_model_2.add_death_flow('death', 1.0 / Parameter('infectious_period'),
 ↪  'infectious')

# Modify deaths and recovery flows according to IFRs
age_strat_2 = get_age_strat(epi_model_2._original_compartment_names)
age_strat_2.set_flow_adjustments('death', {k: Parameter(f'ifr_{k}') for k in
 ↪  age_strata})
age_strat_2.set_flow_adjustments('recovery', {k: 1.0 - Parameter(f'ifr_{k}')
 ↪  for k in age_strata})
epi_model_2.stratify_with(age_strat_2)
for age in age_strata:
    epi_model_2.request_output_for_flow(f'deaths_{age}', 'death',
 ↪  source_strata={'age': age})
```

```
# Specify parameters, run and plot outputs
epi_model_2.run(parameters)
plot_age_deaths(epi_model_2)
```




```
# Compare the total population under either approach
pd.concat([epi_model_1.get_outputs_df().sum(axis=1),
 ↪  epi_model_2.get_outputs_df().sum(axis=1)], axis=1).plot()
```

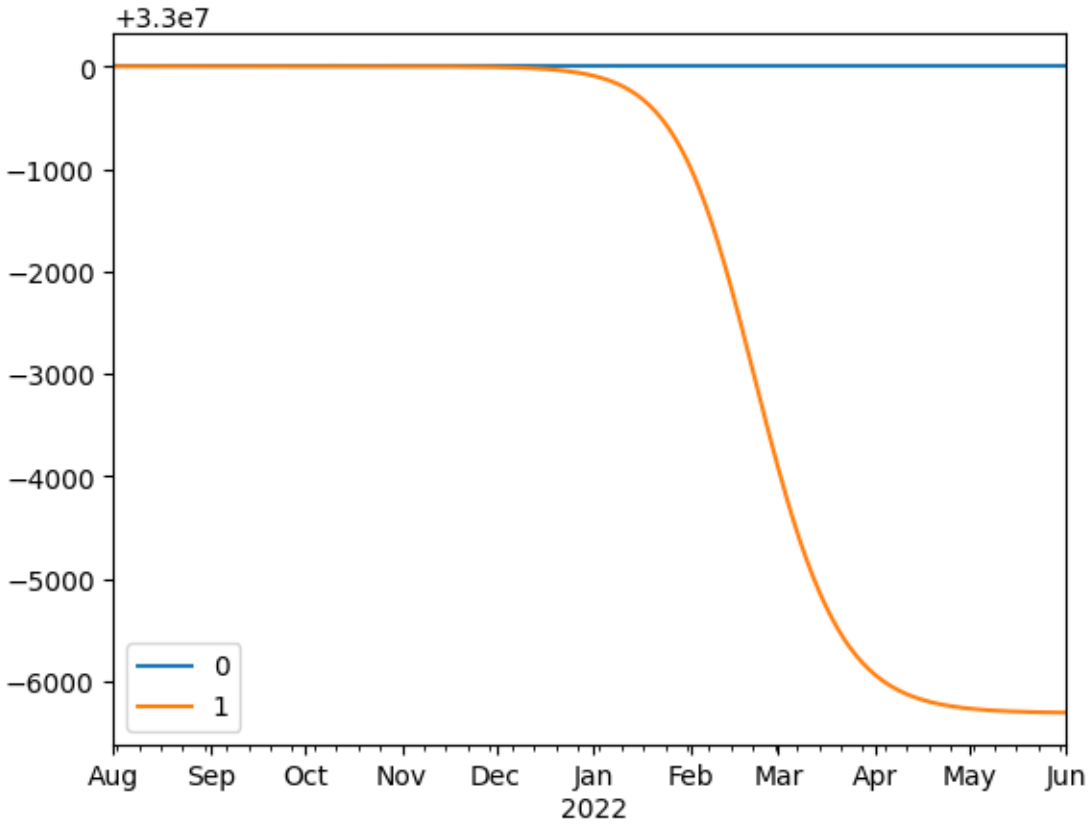


Figure 40: This figure compares the total population over time under two different epidemiological modeling approaches (represented by epi_model_1 and epi_model_2). The plot tracks the sum of the population across different compartments (e.g., Susceptible, Exposed, Infectious, Recovered) to observe how the total population changes over time under each approach.

Multiple strains

```
# If running from Colab, please uncomment and run the following cell to get
 ↪  summer installed.
#!pip install summerepi2==1.3.5
```

```
import pandas as pd
from jax import numpy as jnp
from datetime import datetime
```

```
from summer2 import CompartmentalModel, StrainStratification
from summer2.parameters import Parameter, Function, Time
```

Multiple competing strains

This notebook applies a two-strain stratification to the base model. The two strains compete for the same susceptible population. The assumption that recovered persons are immune to futher infection has not been adjusted.

Seeding

So that the two strains are seeded in the same way, we first ensure that all of the starting population is susceptible. We then use a triangular function to inject infectious persons into the model in order to seed the epidemic with each strain. The time that each strain is introduced is then parameterised for each. The duration and rate of the seed is relatively unimportant because most of the dynamics are driven by transmission as soon as the prevalence of the new strain reaches significant levels.

Competition

Because the assumption of complete immunity after infection has not beeen relaxed, this implies that after recovery, people are fully immune to infection with either strain. Given that we introduce the second (`mutant`) strain some time after the introduction of the first (`wild`) strain when the epidemic has begun to decline, there are relatively fewer susceptibles for the second strain to infect. We therefore increase its infectiousness to very high levels to get an observable epidemic. This is unlikely to be the best way to model most competing strain dynamics, but illustrates the code structure for implementing multi-strain models. In practice, and for simulating COVID-19 dynamics, we would likely wish to allow for subsequent strains to escape past immunity. To achieve this, we would have to implement reinfection processes in the model.

```
analysis_start_date = datetime(2021, 1, 1)
analysis_end_date = datetime(2023, 1, 1)

base_comps = ['susceptible', 'exposed', 'infectious', 'recovered']

epi_model = CompartmentalModel(
    [analysis_start_date, analysis_end_date],
    base_comps,
```

```
    ['infectious'],
    ref_date=datetime(2019, 12, 31),
)
epi_model.add_infection_frequency_flow('infection',
 ↪  Parameter('contact_rate'), 'susceptible', 'exposed')
epi_model.add_transition_flow('progression', 1.0 /
 ↪  Parameter('incubation_period'), 'exposed', 'infectious')
epi_model.add_transition_flow('recovery', 1.0 /
 ↪  Parameter('infectious_period'), 'infectious', 'recovered')
epi_model.set_initial_population({'susceptible': 33e6})
incidence = epi_model.request_output_for_flow('incidence', 'progression',
 ↪  save_results=False)
epi_model.request_function_output('notifications', incidence *
 ↪  Parameter('detection_prop'));
```

```
def triangle_wave_func(
    time: float,
    start: float,
    duration: float,
    peak: float,
) -> float:
    """Generate a peaked triangular wave function
    that starts from and returns to zero.

    Args:
        time: Model time
        start: Time at which wave starts
        duration: Duration of wave
        peak: Peak flow rate for wave

    Returns:
        The wave function
    """
    gradient = peak / (duration * 0.5)
    peak_time = start + duration * 0.5
    time_from_peak = jnp.abs(peak_time - time)
    return jnp.where(time_from_peak < duration * 0.5, peak - time_from_peak *
     ↪  gradient, 0.0)
```

```
# Strain stratification
comps_to_stratify = [c for c in base_comps if c != 'susceptible']
```

```
strain_strata = ['wild', 'mutant']
strain_strat = StrainStratification('strain', strain_strata,
 ↪  comps_to_stratify)
epi_model.stratify_with(strain_strat)
strain_strat.add_infectiousness_adjustments('infectious', {'wild': None,
 ↪  'mutant': Parameter('mutant_infectiousness')})
for strain in strain_strata:
    seed_args = [Time, Parameter(f'{strain}_seed_time'),
 ↪  Parameter('seed_duration'), Parameter('seed_peak')]
    voc_seed_func = Function(triangle_wave_func, seed_args)
    epi_model.add_importation_flow(f'seed_{strain}', voc_seed_func,
 ↪  'infectious', split_imports=True, dest_strata={'strain': strain})
    strain_inc = epi_model.request_output_for_flow(f'incX{strain}',
 ↪  'progression', dest_strata={'strain': strain}, save_results=False)
    epi_model.request_function_output(f'notifX{strain}', strain_inc *
 ↪  Parameter('detection_prop'))

epoch = epi_model.get_epoch()
```

```
parameters = {
    'contact_rate': 0.4,
    'incubation_period': 5.0,
    'infectious_period': 5.0,
    'detection_prop': 0.15,
    'wild_seed_time': epoch.datetime_to_number(datetime(2021, 1, 1)),
    'mutant_seed_time': epoch.datetime_to_number(datetime(2021, 7, 1)),
    'seed_duration': 10.0,
    'seed_peak': 10.0,
    'mutant_infectiousness': 4.0,
}
epi_model.run(parameters)
epi_model.get_derived_outputs_df()[[f'notifX{strat}' for strat in
 ↪  strain_strata]].plot.area()
```

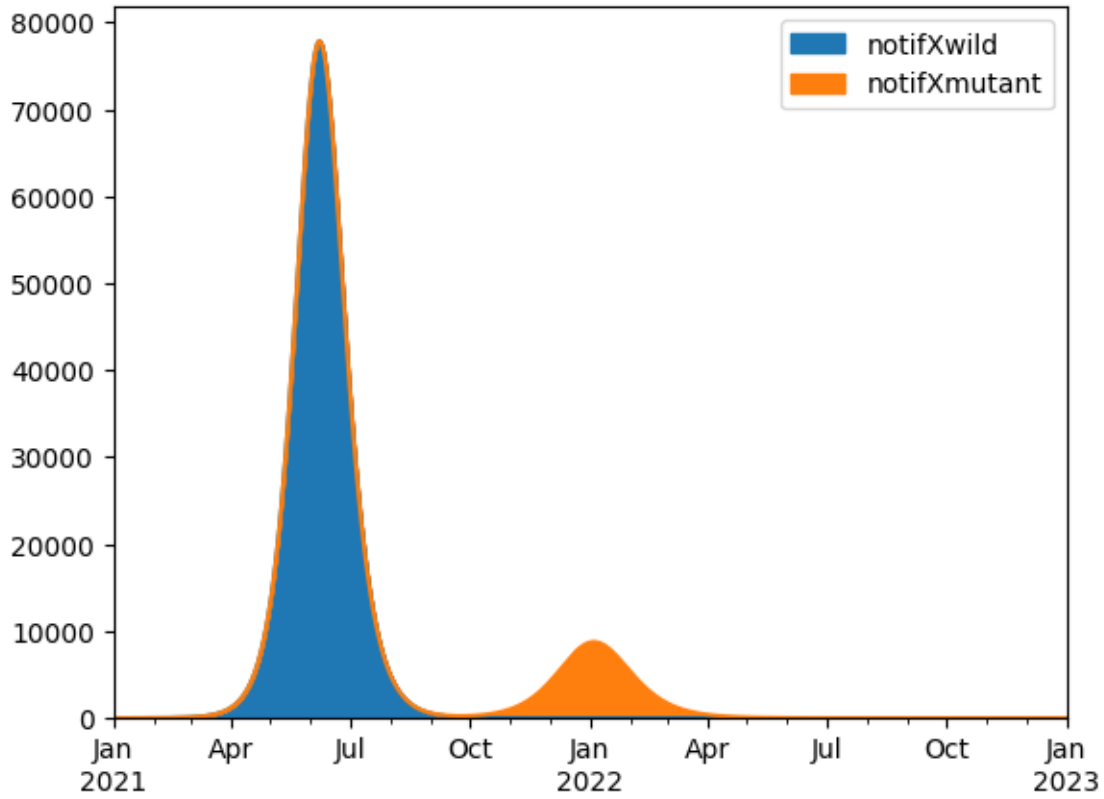


Figure 41: This figure demonstrates the output of an epidemiological model that includes both wild-type and mutant strains of a pathogen. The parameters for the model are specified, and the results are plotted to show the notifications (cases) over time for each strain.

Renewal introduction

A simple renewal model

To get started, we'll implement a renewal model that calculates incidence forward in time but ignores susceptible depletion and a varying reproduction number, such that we will consider:

$$I_t = R_0 \sum_{\tau<t} I_\tau g_{t-\tau}$$

Illustration

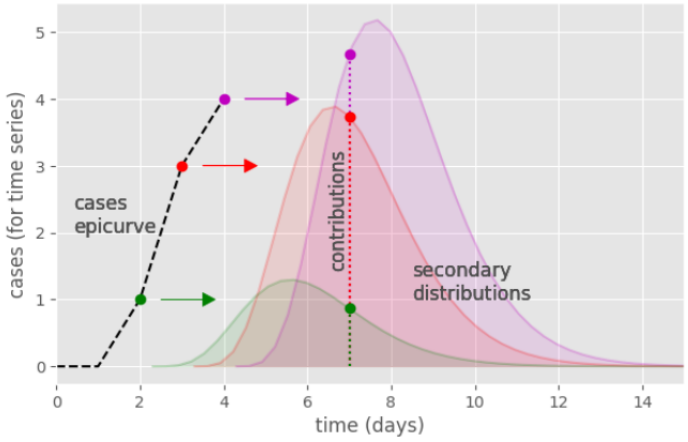


Figure 43: image.png

Figure 42: This figure provides an illustration of the relationship between the cases epicurve, contributions to the epidemic curve, and secondary distributions.

The figure illustrates the concept of how initial cases (epicurve) lead to secondary distributions over time. It visually represents the contributions of individual cases to the overall epidemic curve. Understanding this relationship is crucial for epidemiological modeling, as it helps in forecasting the spread and impact of an epidemic based on initial case data. This illustration aids in comprehending the dynamics of disease spread and the importance of early interventions to curb further transmissions.

Key Components

Cases Epicurve:

- Represented by the dashed black line with green and red points.
- Shows the progression of cases over time.
- Points on the line indicate specific observations at different time points.

Contributions:

- Represented by vertical dashed lines from the cases epicurve to the secondary distributions.

- Points (green, red, and purple) on the cases epicurve are linked to their corresponding secondary distributions.

Secondary Distributions:

- Represented by the colored areas (green, red, purple) under the curve.
- These distributions show the spread of cases over time originating from specific points on the cases epicurve.

Interpretation

Time Series:

- The x-axis represents time in days.
- The y-axis represents the number of cases.

Temporal Dynamics:

- The illustration highlights how initial cases contribute to subsequent secondary distributions.
- Each point on the epicurve (green, red, purple) represents an initial case that generates a secondary distribution over time.
- These secondary distributions illustrate the spread and increase in cases resulting from initial infections.

Epidemic Progression:

- The secondary distributions show how each initial case (on the epicurve) leads to further cases.
- The spread of secondary distributions highlights the compounding effect of initial infections on the overall epidemic curve.
- The overlapping secondary distributions indicate how multiple initial cases contribute simultaneously to the epidemic's progression.

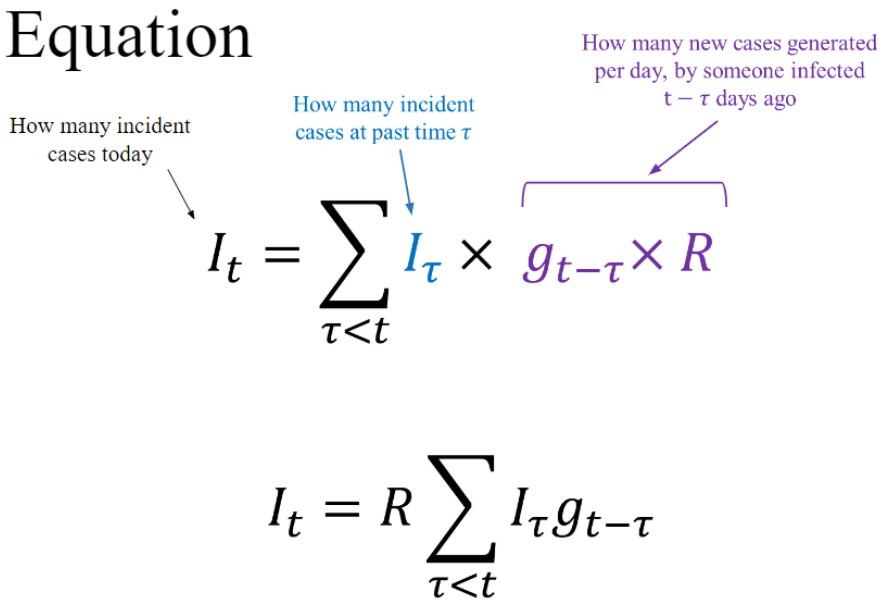


Figure 44: image.png

Figure 43: This figure presents an equation used to calculate the number of incident cases at a given time (I_t) based on the contributions from past incident cases and the generation time distribution.

- This equation models the number of new cases (I_t) at a given time based on the contributions from previous cases (I_τ) and the generation time distribution $(g_{t-\tau})$.
- The reproduction number ( ) is a critical parameter, reflecting how contagious the infection is.
- By summing the contributions of past cases, weighted by the generation time distribution and scaled by , this equation captures the dynamics of disease spread.
- This formula is fundamental in epidemiology for predicting the future trajectory of an epidemic and understanding the transmission dynamics.

```
from scipy.stats import gamma
import numpy as np
import pandas as pd
```

Generation time

We'll get a distribution we can sensibly use for the generation time, which could represent an acute immunising respiratory infection (assuming the time unit is days).

```
# Generation time summary statistics
gen_mean = 5.0
gen_sd = 1.5

# Calculate equivalent parameters
var = gen_sd ** 2.0
scale = var / gen_mean
a = gen_mean / scale
gamma_params = {"a": a, "scale": scale}

# Get the increment in the CDF
# (i.e. the integral over the increment by one in the distribution)
gen_time_densities = np.diff(gamma.cdf(range(1024), **gamma_params))

pd.Series(gen_time_densities).iloc[0:20].plot()
```

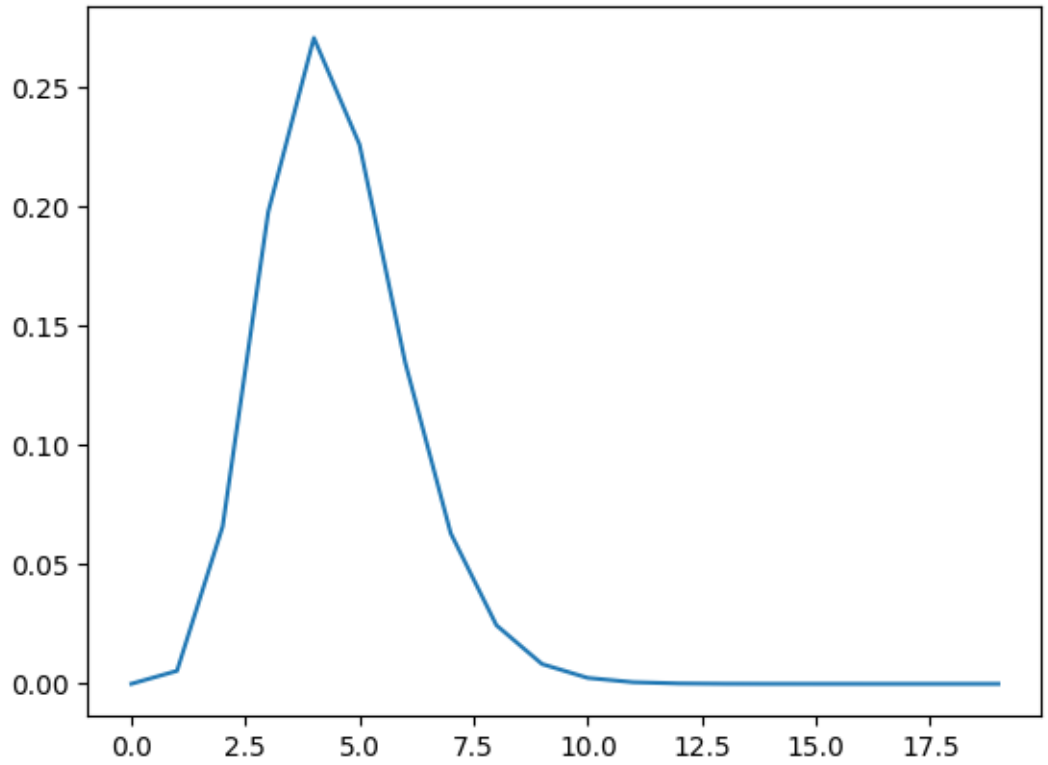


Figure 34: This figure shows the generation time distribution used for modeling an acute immunising respiratory infection, assuming the time unit is days.

Generation Time Summary Statistics:

- The mean generation time is 5.0 days.
- The standard deviation is 1.5 days.

Gamma Distribution Parameters:

- The shape and scale parameters of the gamma distribution are calculated to match the given mean and standard deviation.
- The gamma distribution is used to represent the generation time, which describes the time intervals between successive cases in a chain of transmission.

Density Plot:

- The plot shows the distribution of generation times, indicating how the likelihood of secondary cases varies over time.
- The density initially increases, peaking around 5 days, and then decreases, reflecting the spread of secondary cases from an initial infection.

The generation time distribution is crucial for modeling the dynamics of infectious diseases. By using the mean and standard deviation, the gamma distribution parameters are calculated to create a realistic model of the generation time. The density plot helps visualize how the likelihood of secondary cases changes over time, which is essential for predicting the spread and impact of an epidemic. Understanding the generation time distribution aids in accurate epidemiological modeling and helps inform public health interventions.

Calculations

Here, we'll start with naive Python loops with pre-calculated generation times to be completely explicit (but slow). Note that the delay is specified as `t - tau - 1` because delay then starts from zero each time, which then indexes the first element of the generation time densities. As shown in the previous cell, the `gen_time_densities` is the integral of the probability density over each one-unit interval of the gamma distribution.

```
# Let's set some arbitrary parameters to start with
n_times = 25
seed = 1.0
r0 = 2.0
incidence = np.zeros(n_times)
incidence[0] = seed
```

```
for t in range(1, n_times):
    val = 0.0
    for tau in range(t):  # For each day preceding the day of interest
        delay = t - tau - 1  # The generation time index for each preceding
↪   day to the day of interest
        val += incidence[tau] * gen_time_densities[delay] * r0  # Calculate
↪   the incidence value
    incidence[t] = val
```

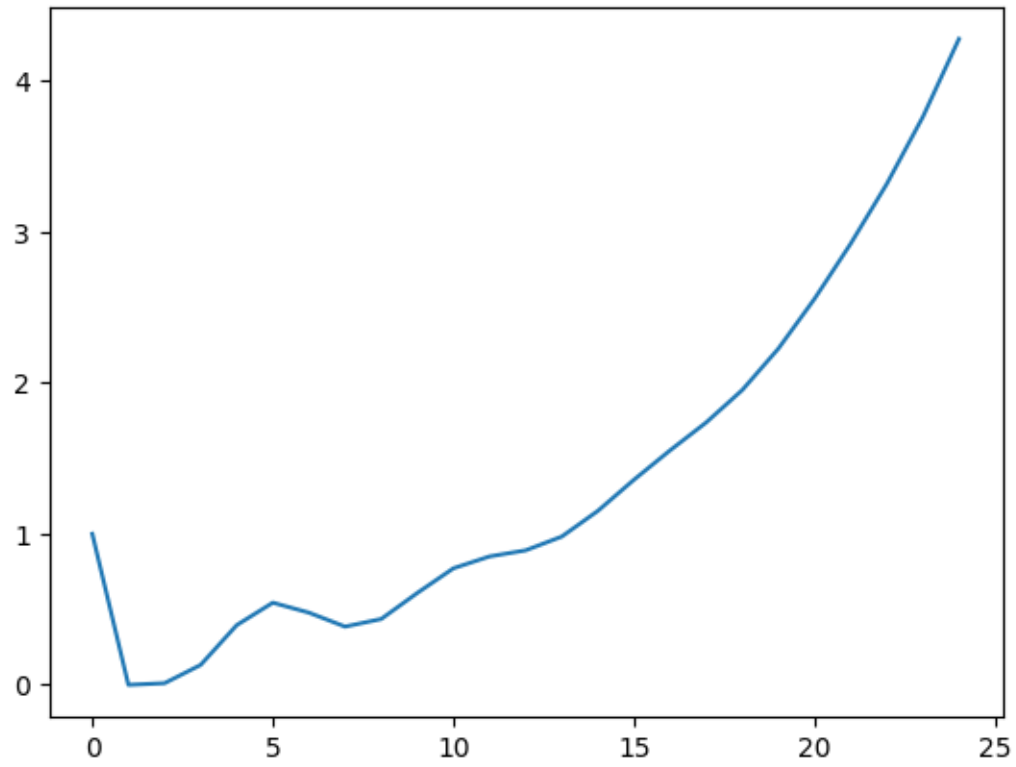
We can get this down to a one-liner if preferred. The epidemic is going to just keep going up exponentially, of course, because $R_0 > 1$ and there is no susceptible depletion.

```
alternative_inc = np.zeros(n_times)
alternative_inc[0] = seed

for t in range(1, n_times):
    alternative_inc[t] = (alternative_inc[:t] *
↪   gen_time_densities[:t][::-1]).sum() * r0
```

```
np.allclose(incidence, alternative_inc) # Check our 2 methods are the same

pd.Series(incidence).plot()
```



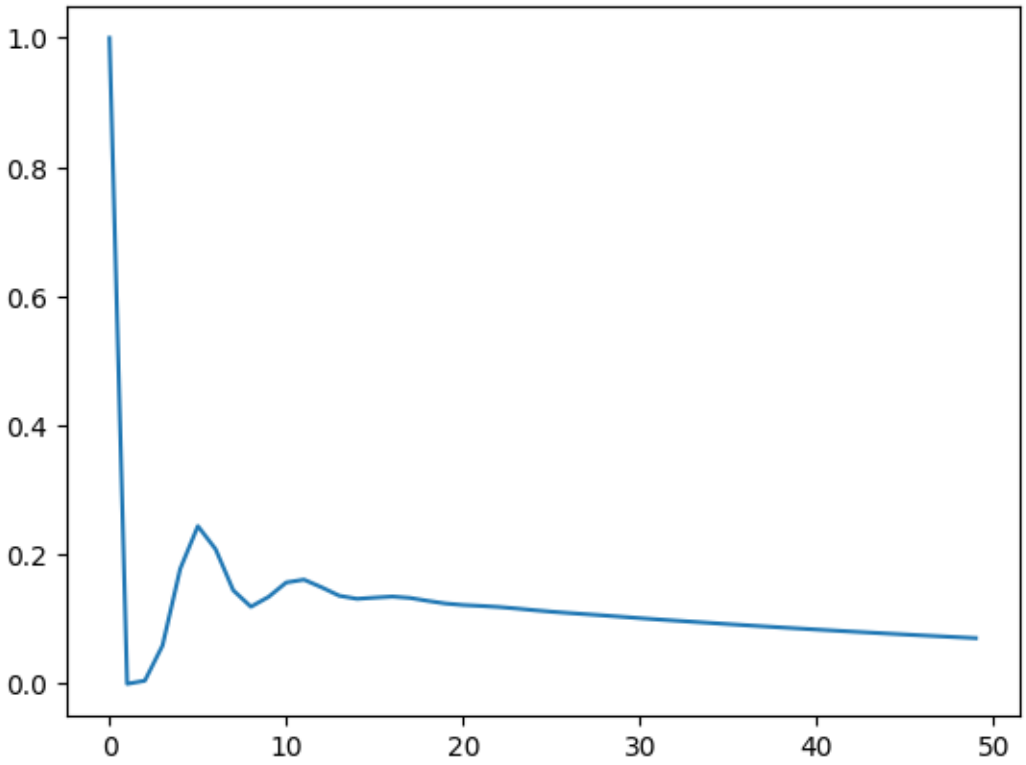
Already some interesting phenomena are emerging, in that the humps are the generations of cases from the first seeding infection (which occurs at a single time point), which progressively smooth into one-another with generations of cases.

Threshold behaviour

We expect a declining epidemic with $R_0 < 1$, and equilibrium at $R_0 = 1$

```
n_times = 50
low_r_inc = np.zeros(n_times)
low_r_inc[0] = 1.0
# Try changing the r0 value
r0 = 0.9
for t in range(1, n_times):
```

```
    low_r_inc[t] = (low_r_inc[:t] * gen_time_densities[:t][::-1]).sum() * r0
pd.Series(low_r_inc).plot()
```



Susceptible depletion

To add one layer of realism, we'll now start to think about susceptible depletion, considering the equation:
$I_t = (1 - \frac{n_t}{N}) R_0 \sum_{\tau<t} I_\tau g_{t-\tau}$

We'll now run the model with susceptible depletion, decrementing the susceptible population by the incidence at each step. We'll also zero out any negative values for the susceptibles that could occur if the time step is too large (which should be negligible for reasonable time step and parameter choices). We'll need a higher reproduction number to deplete the susceptible population within the time window we have.

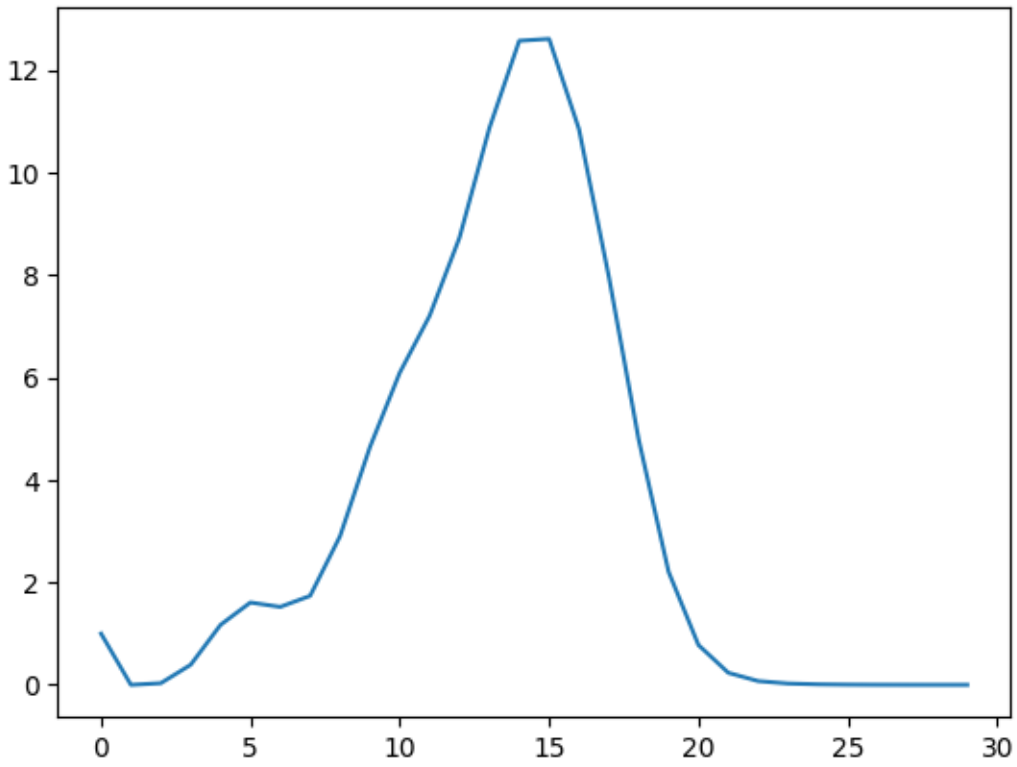
```
n_times = 30

r0 = 6.0
```

```
pop = 100.0
deplete_inc = np.zeros(n_times)
deplete_inc[0] = seed
suscept = pop - seed

for t in range(1, n_times):
    suscept_prop = suscept / pop
    infect_contribution_by_day = deplete_inc[:t] *
 ↪  gen_time_densities[:t][::-1] * r0
    this_inc = infect_contribution_by_day.sum() * suscept_prop
    deplete_inc[t] = this_inc
    suscept = max(suscept - this_inc, 0.0)

pd.Series(deplete_inc).plot()
```



Now with susceptible depletion, we have an epi-curve that goes up in the initial phase with $R_0 > 1$, but comes back down as susceptibles are depleted and so R_t falls below one.

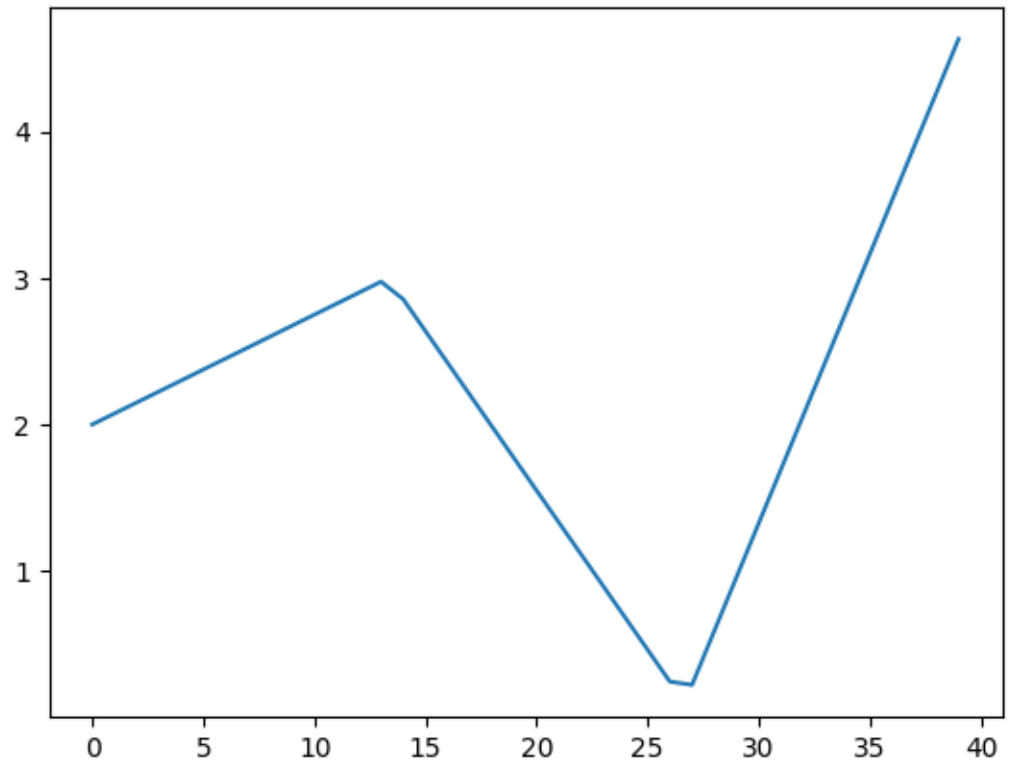
Varying the reproduction number

Building on the previous cells and including susceptible depletion, we'll now look at varying the reproduction number with time, because inferring the variation in this quantity is what we're aiming to achieve from these models.

As previously, the equation we're considering will be:
$I_t = (1 - \frac{n_t}{N}) R_t \sum_{\tau<t} I_\tau g_{t-\tau}$ However, now the R_t value is determined both by an extrinsic variable ("random") process. At this stage, the process will be arbitrary values, and there are several functions that could be used (including a random walk and an autoregressive process).

First, let's set up a variable process, which we'll define as the relative variation in the reproduction number as time proceeds.

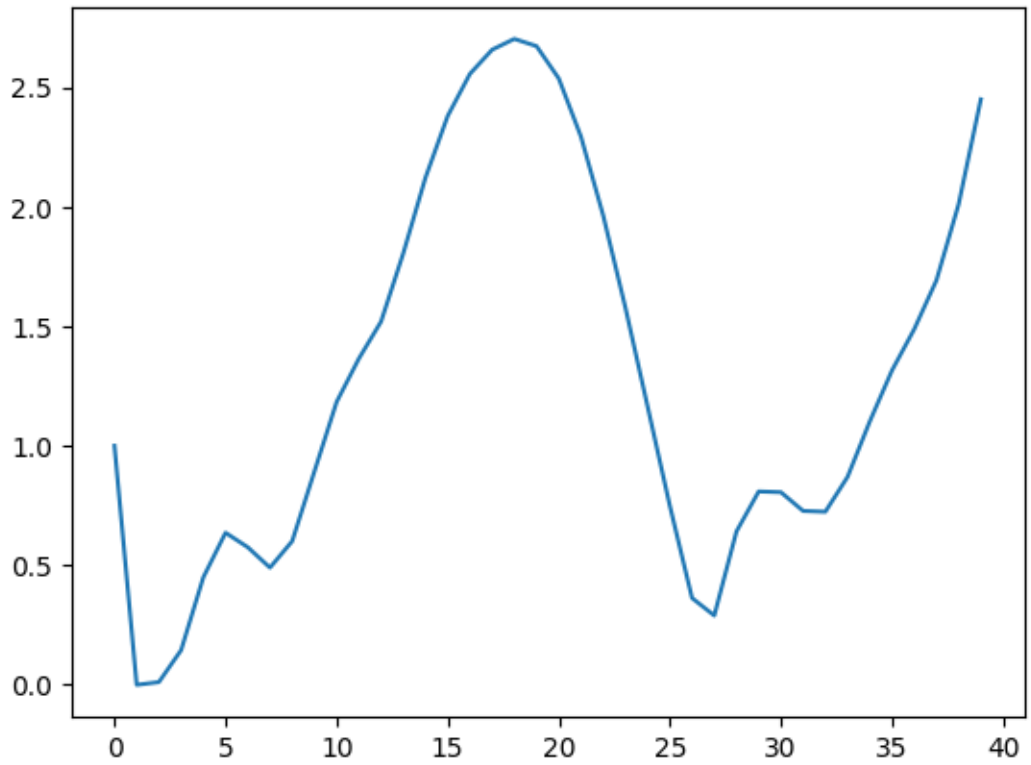
```
n_times = 40
process_req = [2.0, 3.0, 0.1, 5.0]
process_times = np.linspace(0.0, n_times, len(process_req))
process_vals = np.interp(range(n_times), process_times, process_req)
pd.Series(process_vals, index=range(n_times)).plot()
```



We'll use model parameters, population size and the generation times as previously, and run the model with both susceptible depletion, and the variable process. Now we can manipulate the shape of the epicurve. The evolution of the variable process is what we'll estimate in later sessions to estimate the variation in the reproduction number over time.

```
var_r_inc = np.zeros(n_times)
var_r_inc[0] = seed
r0 = 1.0

suscept = pop - seed
for t in range(1, n_times):
    suscept_prop = suscept / pop
    infect_contribution_by_day = var_r_inc[:t] * gen_time_densities[:t][::-1]
 ↪  * r0
    this_inc = infect_contribution_by_day.sum() * suscept_prop *
 ↪  process_vals[t]
    var_r_inc[t] = this_inc
    suscept = max(suscept - this_inc, 0.0)
pd.Series(var_r_inc).plot()
```



Other resources

The notebooks listed above provide a brief introduction to participants in our 2023/2024 capacity building program.

Further resources are available through our main homepage. In particular, we draw your attention to:

- Full online documentation of the summer platform
- Our textbook of infectious disease modelling using summer

A Textbook of Infectious Diseases Modelling using the summer Platform

This textbook is maintained by the Epidemiological Modelling Unit at Monash University's School of Public Health and Preventive Medicine. The notebooks (or chapters) run over Google's Colab interface, and are introduced in the first notebook below. Of course, we encourage users to download these and run them locally with their preferred environments and interfaces. Feedback is very welcome, please send comments to james.trauer@monash.edu

Table of Contents

- Notebook 13 Frequency-dependent and density-dependent transmission with heterogeneous mixing
- Notebook 14 Assortative mixing
- Notebook 15The relationship between mixing matrices and heterogeneity in susceptibility and infectiousness
- Notebook 16 Using empiric data on population interactions in our models (introduction)
- Notebook 17Understanding data from contact surveys
- Notebook 18 Implementing contact survey data in a model
- Notebook 19 Adapting mixing matrices to new contexts
- Notebook 20 Model calibration with the Metropolis algorithm