yo

Abstract

uadnf+ads

1 Introduction

blalba

2 Theoretical Background

3 Algorithms, implementation and testing

As this project is new this year (as far as I know), some trial and error, as well as some creative freedom to change equations for more realistic implementation was taken.

3.1 Runge-Kutta model

The Runge-Kutta method is a method similar to Euler's method for solving ODEs, however it provides an intermediate step in between the steps, which leads to better results. To understand this, we begin with the equation

$$\frac{dy}{dt} = f(t, y)$$

where

$$y(t) = \int f(t, y)dt$$

and

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y)dt$$

The final algorithm is

$$\int_{t_i}^{t_{i+1}} f(t, y)dt \approx \frac{h}{6}\left[f\left(t_i, y_i\right) + 4f\left(t_{i+1/2}, y_{i+1/2}\right) + f\left(t_{i+1}, y_{i+1}\right)\right] + O\left(h^5\right)$$

so that

$$y_{i+1} \approx y_i + \frac{h}{6}\left[f\left(t_i, y_i\right) + 2f\left(t_{i+1/2}, y_{i+1/2}\right) + 2f\left(t_{i+1/2}, y_{i+1/2}\right) + f\left(t_{i+1}, y_{i+1}\right)\right]$$

To implement this it is easier to consider each step as k_i, $i \in \{1, 2, 3, 4\}$. The final algorithm becomes:

$$k_1 = hf\left(t_i, y_i\right)$$

$$k_2 = hf\left(t_i + h/2, y_i + k_1/2\right)$$

$$k_3 = hf\left(t_i + h/2, y_i + k_2/2\right)$$

$$k_4 = hf\left(t_i + h, y_i + k_3\right)$$

$$y_{i+1} = y_i + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

In our case, where f is a function $f(t, S, I, R)$, we define Sk_i, Ik_i and Rk_i where $i \in \{1, 2, 3, 4\}$. So that

$$Sk_1 = hf_S(t_i, S_i, I_i, R_i)$$

and

$$Sk_2 = hf_S(t_i + 0.5h, S_i + 0.5Sk_1, I_i + 0.5Ik_1, R_i + 0.5Rk_1)$$

and so forth until finally

$$S_{i+1} = S_i + \frac{1}{6}(Sk_1 + 2Sk_2 + 2Sk_3 + Sk_4)$$

This will be the same for I and R.

To apply different models for S, I and R, the algorithm just changes the function for f. Before every run, a pointer to which member function to use for S, I and R is assigned. These member functions take in different input parameters, implementing vital parameters, seasonal variations and vaccines. By using pointers to functions, the actual RK-solver only has to be written once, where the function f is easily changed for different runs.

3.2 Monte Carlo method

In the Monte Carlo method, the approach taken here is that for every timestep, there is a chance that a member in the population can change state. These probailities, when the timestep is small enough so that only, on average, one such chang ecan happen per timestep, can be considred a transition probability. Most of the transition probabilites are independent of the amount of people in the population, while some depend on things in the population such as cahnce of infection whcih depends on the amount of infected people.

To manage the population, a class Person is used, which does a few simple things:

1. Holds the state of a member of the population (0 = Suspeptible, 1 = Infected, 2 = Recovering)
2. Has simple get and set functions for the state

For each timestep, or Monte Carlo cycle, each person in the population has a transition probability to change state depending on iputs as seen in *Table* 1

Transition	Value	Note
S to I	$a\frac{I_{total}}{N}\Delta t$	Getting infected
S to R	$f\Delta t$	Getting vaccine
I to R	$b\Delta t$	Recovering
R to I	$c\Delta t$	Losing immunity
Any to birth	$e\Delta t$	Giving birth
Any to dead	$d\Delta t$	Natural death
I to dead	$d_I\Delta t$	Dying from infection

Table 1: Transition probabilities in Monte Carlo method

The timestep is defined as

$$\Delta t = \min\left\{\frac{4}{aN}, \frac{1}{bN}, \frac{1}{cN}\right\}$$

Ensuring approximately one transition per timestep. If there are too many transitions per timestep, then the dynamic transition probability such as S to I won't have time to adjust between transitions, so that I_{total} and N aren't the corrent values.

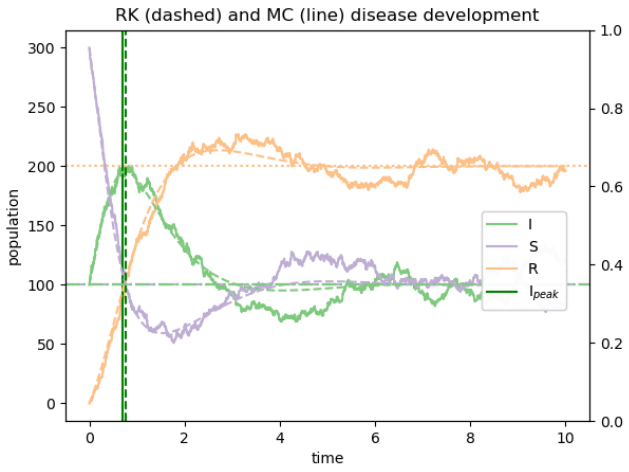


Figure 1: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with $a = 4$, $b = 1$ and $c = 0.5$

3.3 Tests

As the models are best understood visually by looking at the graphs produced, any failsafe way to ensure that every run produces meaningful results can be hard. However, there are expected values for cases where only a, b and c are considered, in which the results can be compared to. Additionally, comparing the RK ODE solver's results with the MC solver's results can give some indication as to whether the program is working, given that these independent methods produce the same results.

3.3.1 Testing the RK solver with expected values

As the MC solver has quite a bit of variance after reaching the steady state, a numeric test for the RK solver is more feasable. By requiring that the final value for lets say I is equal to the expected value within some error ϵ, we can test against the expected values. The expected values are also plotted as horizontal dashed lines when the total population is static as seen in *Figure* 1.

3.3.2 Testing the population manager in the MC solver

As mentioned already, due to the variance in the MC solver, comparing its final state to some expected value isn't really feasable. Therefore, unit tests are checking that the population manager, which ads, removes and manages the states of members of the population, is funcitoning properly. With this unit test passed, and with visual comparison of the RK solver and the MC simulations' plots, results are more trustworthy.

3.4 Optimizations and runtimes

While the RK solver is an algorithm already developed by others, it can't really be optimized much, as it can't run in paralell processes, or simplified much more, the MC solver probably can be optimized a lot. Every person in the population has a lot of if statements to check if the RNG is within the transition probabilities. I am not familiar of any other way to check for RNG¡P, and as such if statements are significantly slowing down the simulations.

Method	MC Cycles/steps	Population	Runtime
RK (2EQ)	2399	400	0.00015s
RK (3EQ)	2399	400	0.00018s
RK (2EQ)	100000	400	0.0069s
RK (3EQ)	100000	400	0.0072s
MC (no OpenMP)	2399	400	0.083s
MC (OpenMP)	2399	400	0.31s

Table 2: A comparison of runtimes between the RK solver and the MC solver.

In *Table* 2, the RK solver is significantly faster than the MC solver. Simplifying from 3 equations to 2 in the RK solver also seems to reduce runtimes somewhat, but not signifficantly. This is likely due to the MC solver doing logic on every single member of the population, while the RK solver calculates the whole population at once. To speed this up, it was attempted to implement OpenMP on the cycle throught the population in the MC solver, however it seemed to create silly results and run slower, so this was not attempted to implement properly.

3.4.1 2 vs 3 equations in RK solver

As metioned briefly already, in order to speed up the RK solver a bit, it checks whether it can simplify from 3 to 2 equatins. When the population is stable, such that N is constant, we have that

$$N = I + R + S \tag{1}$$

Which can be simplified such that

$$R = N - R - S$$

So that R is plugged into the equation for S, and we only need 2 equations. However, as is the case with the vital parameters where N is changing, we now have to update N and R, which depend on each other, which is no longer possible. This version of the RK-model turns out to be slightly slower, as R now needs to be calculated for every step.

3.4.2 Using array of functions to avoid if-statements in MC solver

Like described already, different inputs leads to different transitions being evaluated in the MC solver. In an effort to reduce the amount of if-statements, the pointers to the relevant transition functions evaluating a potential state change are put in an array. In the array, there is one function per state, and as such, the integer representing the state of the currently evaluated member of the population is used to index the array to call the corresponding function. As such, only the get state function which returns an integer is called from the Person class, and an if-statement is avoided.

3.4.3 Compiler flag optimizations

To further increase performance of the program, OpenMP is implemented for consecutive runs of the MC simulation, used to gather statistics for the time of infection peak and when zero infections is reached. Additionally compiler flag optimization -O is compared to -O3. In *Table* 3, we see signifficant performance increase in both MC and RK for -O3, and for OpenMP in MC.

Method	Optimizations	MC Cycles/steps	Consecutive runs	Population	Runtime
RK	None	10 000	1	400	0.0018s
RK	-O3	10 000	1	400	0.00083s
MC	None	2399	20	400	17.83s
MC	-O3	2399	20	400	1.65s
MC	-O3 OpenMP	2399	20	400	0.38s

Table 3: Runtimes with different compiler flag opimizations for single run RK and multiple run MC.

4 Analysis

Here, first a statistical approach is taken to assess the accuracy of the MC model, before further testing different variations of inputs. One thing to note, is that the data shortly after the indection dies out is cut for the MC solver. This is because ashortly fter the infection dies, the whole poulation becomes susceptible with no odds of getting infected. In the RK solver however, infections never reach exactly zero, they only converge to the expected values, which makes this type of determinaiton less viable for the RK solver. The RK solver may dip very close to zero infections, and then rise back up to some higher value. But, since the MC solver is discrete it infections, when 1 turns to 0, nothing more can happen. Thus, if the MC data seem to cutoff early, it is only because nothing of interest happens after that point.

Another thing to note is that there are 15 inputs parameters that can affect the result of the model. So, it is absolutely not in the scope of this report to thoroughly check how every combination works. There are probably combinations that should've or could've been tried, but to keep the report brief and concise, only some combinations and variations of each input parameter group is explored. The main aim of this project is to develop two models for disease, and check that it can handle different inputs, not to map every single combination of inputs possible.

4.1 Error and accuracy

Before looking at any results, it is worth considering the error, and standard deviation in the results we see in the RK and MC solver. The RK solve has error $O(h^5)$, whcich isn't really interesting to investigate further. The MC solver however, does not have a pre-defined theoretical error, and as a result must be evaluated statistically.

In order to make results somewhat comparable, an attempt to keep the parameters the same as much as possible is made.

In *Figure* 2 and 3, a simulation is ran with $a = 4$, $b = 4$ and $c = 0.5$ 500 times, where we expect an initial peak of infections, before the infection dies out completely. However, we see that the time of the infection peak, as well as the time of the infection being over, has σ within the same order of magnitude as the mean. This means that although the MC solver roughly does what you'd expect, there's a lot of random variety within each set of initial conditions.

In *Figure* 4, the mean value of the number of infections after reaching a steady state is calculated, and again we see significant variation. However, compared to infection peak and infection over as discussed above, the σ value is roughly one order of magnitude lower than the mean.

In general, the MC solver has a lot of variety, yet it seems to do what it should within an order of magnitude of time. The variation after reaching the steady state seems be much lower, while still having quite a bit of random variation.

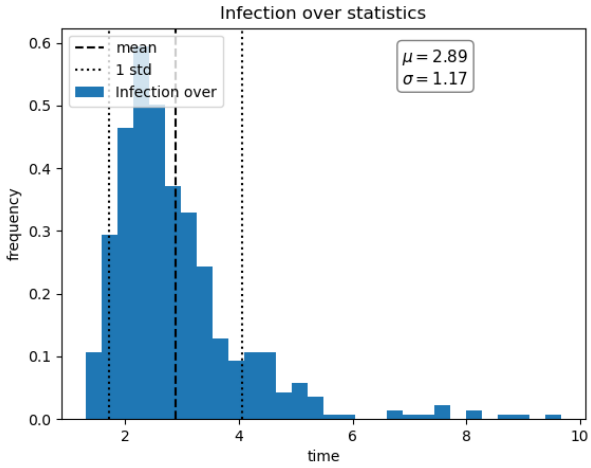


Figure 2: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with $a = 4$, $b = 4$ and $c = 0.5$

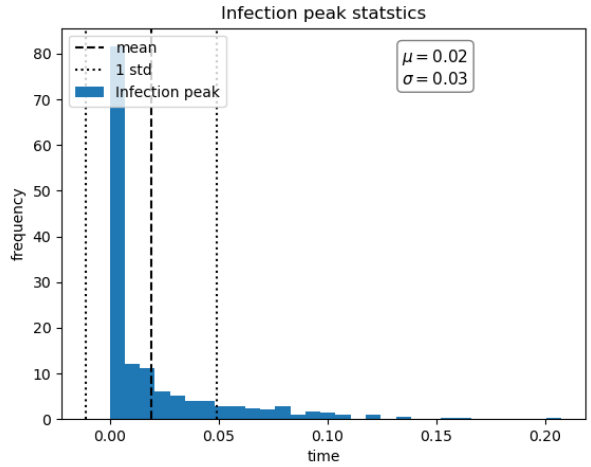


Figure 3: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with $a = 4$, $b = 4$ and $c = 0.5$

4.2 The impact of changing b

The first scenario to be considered, is with only a, b and c. To keep things simple, only b will change such that $b \in \{1, 2, 3, 4\}$. From the equations for expected values, we need $b < a$ in order for an infection to reach a steady state within a population. In *Figure* 5, the RK and MC approach the expected values, and we see that the infection manages to reach steady state within the population for $b = 1$ and $b = 2$. However, due to the randomness in the MC solver, as discussed above, the infection actually dies out in the run with $b = 3$, due to the expected amount of infected people only being 14 at the steady state. With a much larger population, this is expected to be less likely to happen. For $b = 4$, the infection dies out as expected both for the MC and RK solver.

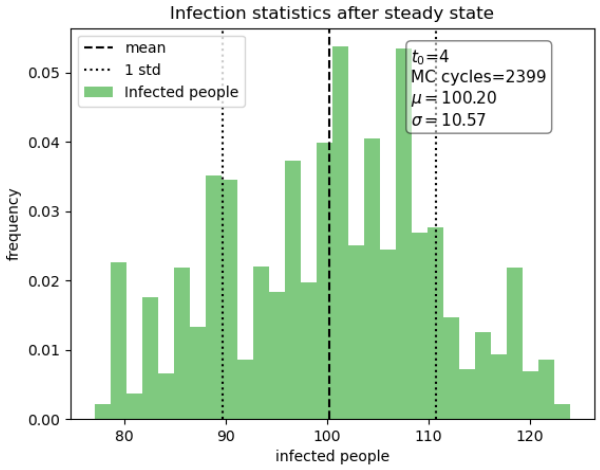


Figure 4: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with $a = 4$, $b = 4$ and $c = 0.5$

4.3 Vital Parameters

The next scenatio to be considered is including vital parameters narutal death d, natural birth e and death due to disease d_I. Having a ridiculously high birth rate or natural death rate doesn't really make any sense here, as it would either kill everyone or explode the population. Ideally we want the population to be somewhat stable, and see how disease deaths influence the steady state of the system. The key aspect here, is that when people die from the infection, it keeps them from infecting others, essentially killing the disease by removing carriers of the disease. To explore this further, $e = 0.011$ and $d = 0.01$, while d_I varies. Some key plots will be included to show how d_I affects the steady state, and how fast the infection is over.

In *Figure* 6, varying values of d_I are shown for the previously stable case where $b = 2$. However, with vital parameters introduced, we see that the disease ceases to be stable within the population, and eventually all the carriers are dead in the MC solver. In the RK solver however, all the values converge to zero for $d_I = 0.05$ and $d_I = 0.1$, in contrast to the MC solver where the disease disappears. In (d), we see that the MC simulation consistantly gets rid of the disease at approximately time = 20.

To summarize, introducing disease deaths to a small population in the MC solver, causes the disease to go extinct faster, while also killing a lot of people. However in the RK solver, the overall deathrate vs birthrate seems to dominate wether the population increases or converges to zero or some stable value. In this case the RK solver is not really useful, as it seems to model disease porrly, and doesn't really model when there are no more infected left well.

In the case of the MC solver, having d_I high actually causes the disease to die faster. To test this further, a smaller amount of initial infections are tested, to see if a high death rate actually prevents a disease to spread. In *Figure* 7 the initial infections are set to 20, but the disease development is still the same, just with a lower peak than before. This is due to the amount of susceptible people being so much higher when the infections are low, causing infections to grow even when the infection death rate is pretty high. So, the death rate does not really seem to determine whether an infetion will die "early", only how fast it might make the disease extinct due to carriers dying after the infection peak.

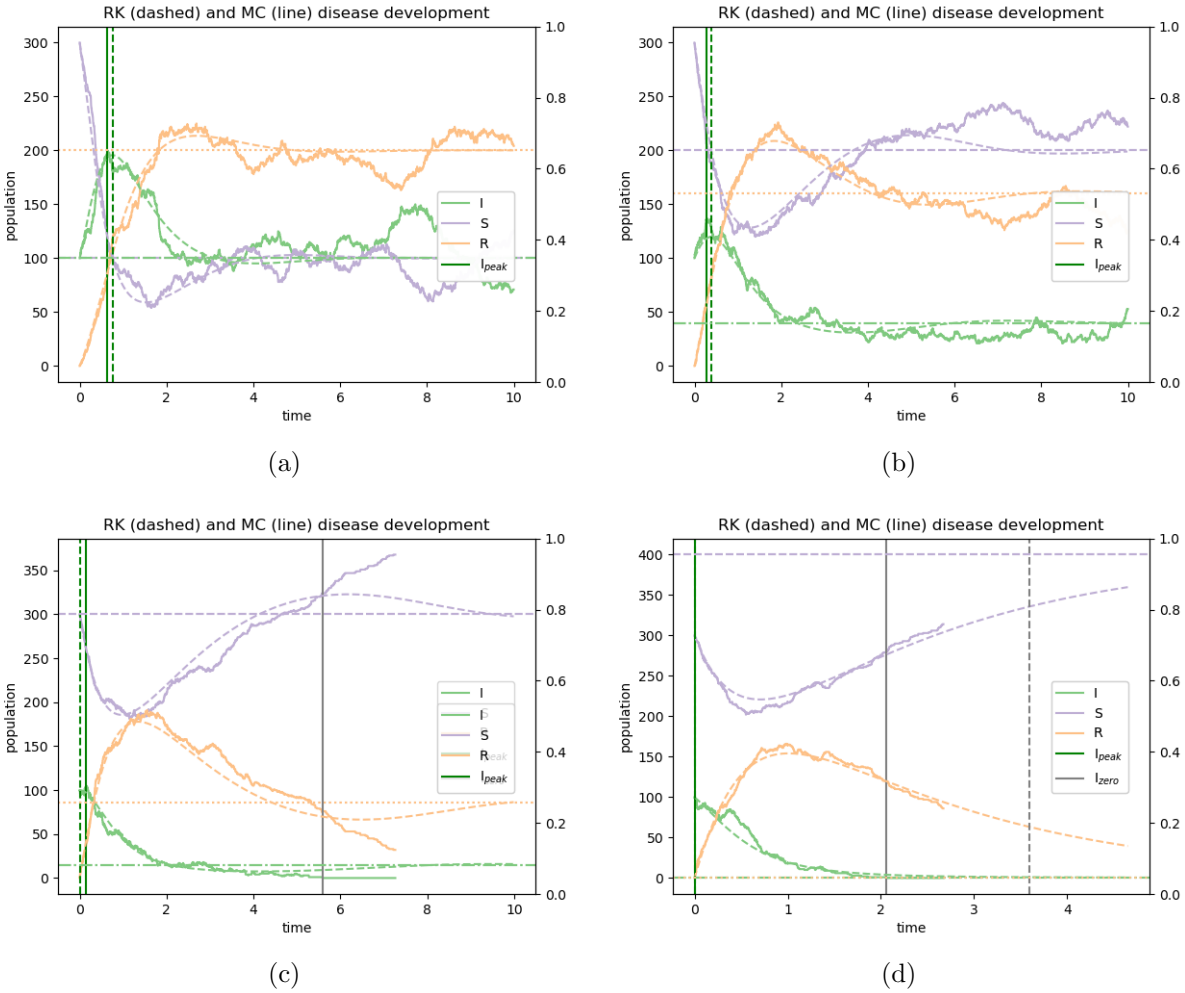


Figure 5: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with fixed $a = 4$ and $c = 0.5$. b has values 1(a), 2(b), 3(c) and 4(d). The horizontal dashed lines are the expected values.

4.4 Seasonal Variation

Another parameter that impacts things like the flu in particular, is seasonal variations. To simulate this, the rate of infection $a_{seasonal}$ is modyfied by

$$a_{seasonal} = a + Acos(\omega t)$$

The average value of a over a period ωk, $k \in \mathbf{N}^+$ remains unchanged, so does this mean that the amount of infected people also remain the same? To explore this, $\omega = 2\pi$, and $A \in \{1, 2, 3, 4\}$ for $a = 4$, $b = 2$ or $b = 1$ and $c = 0.5$. Initially, no vital parameters will be introduced.

Changing b so that $b = 1$ causes the expected values to for I and S to be the same, highlighting the cyclic relationship of the two, as seen in In *Figure* 10

So how does this affect the statistics of when an infection is over, or the mean amount of infected people?

s

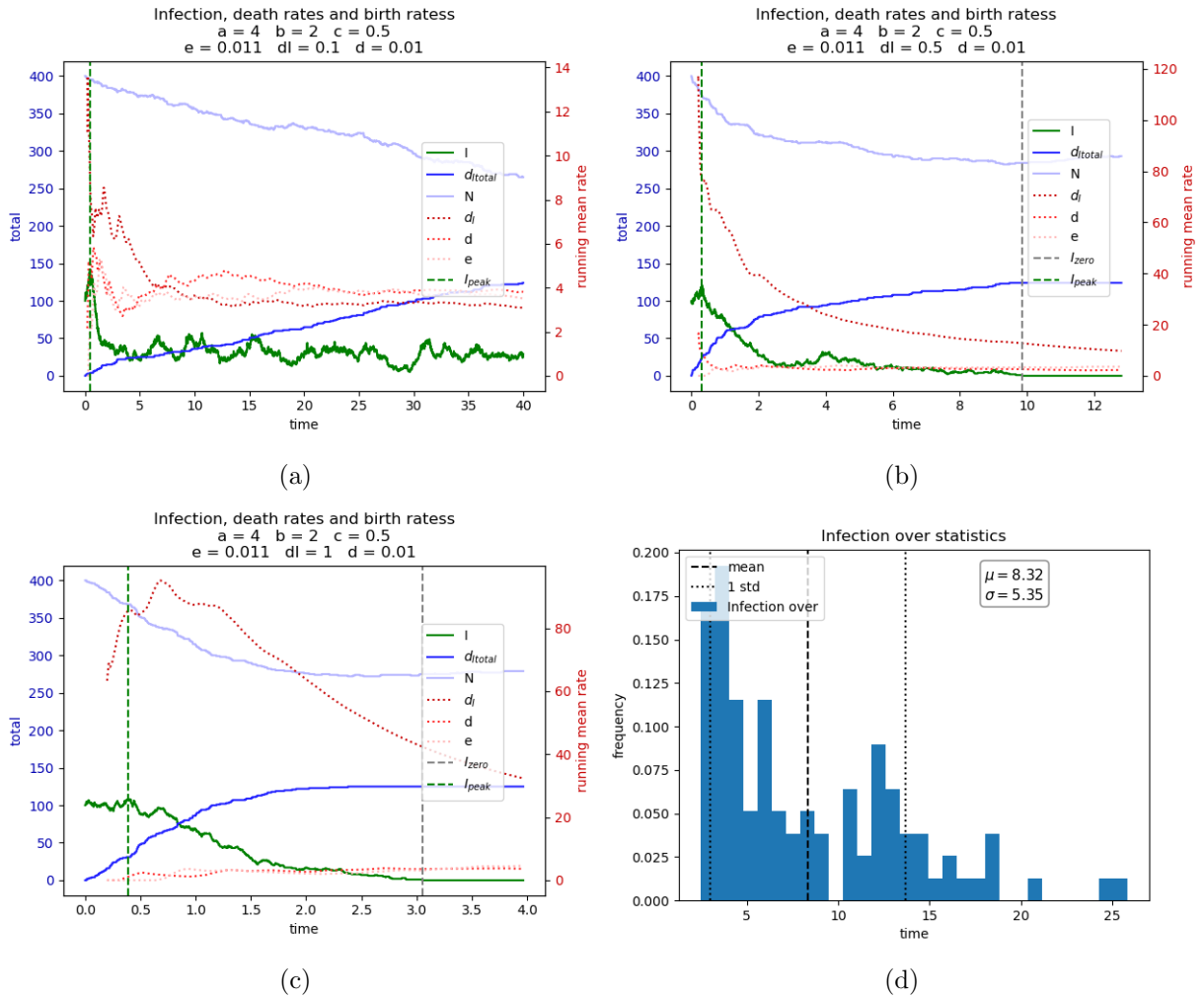


Figure 6: Disease development for $a = 4$, $b = 2$, $c = 0.5$, $e = 0.011$ and $d = 0.01$. d_I varies as 0.1 (a), 0.5(b) and 1(c and d), with 100 runs collecting the statistics in (d)

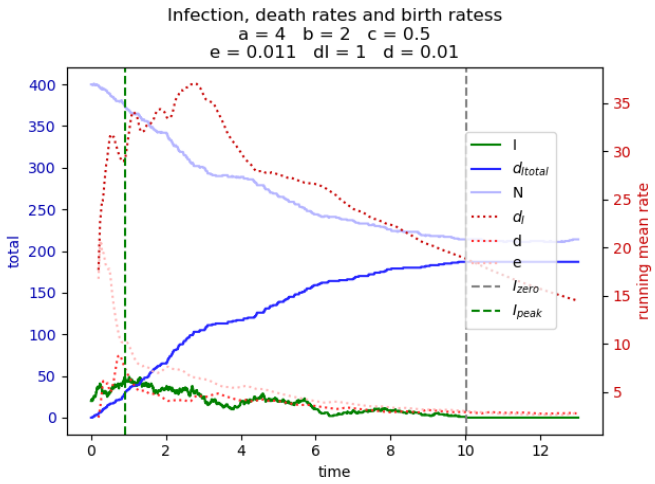


Figure 7: A run with initial infections = 20, with $a = 4$, $b = 2$, $c = 0.5$, $e = 0.011$, $d = 0.01$ and $d_I = 1$.

4.5 Vaccines

5 Discussion

5.1 Accuracy of models

5.2 MC vs RK, strengths and weaknesses

6 Conclusion

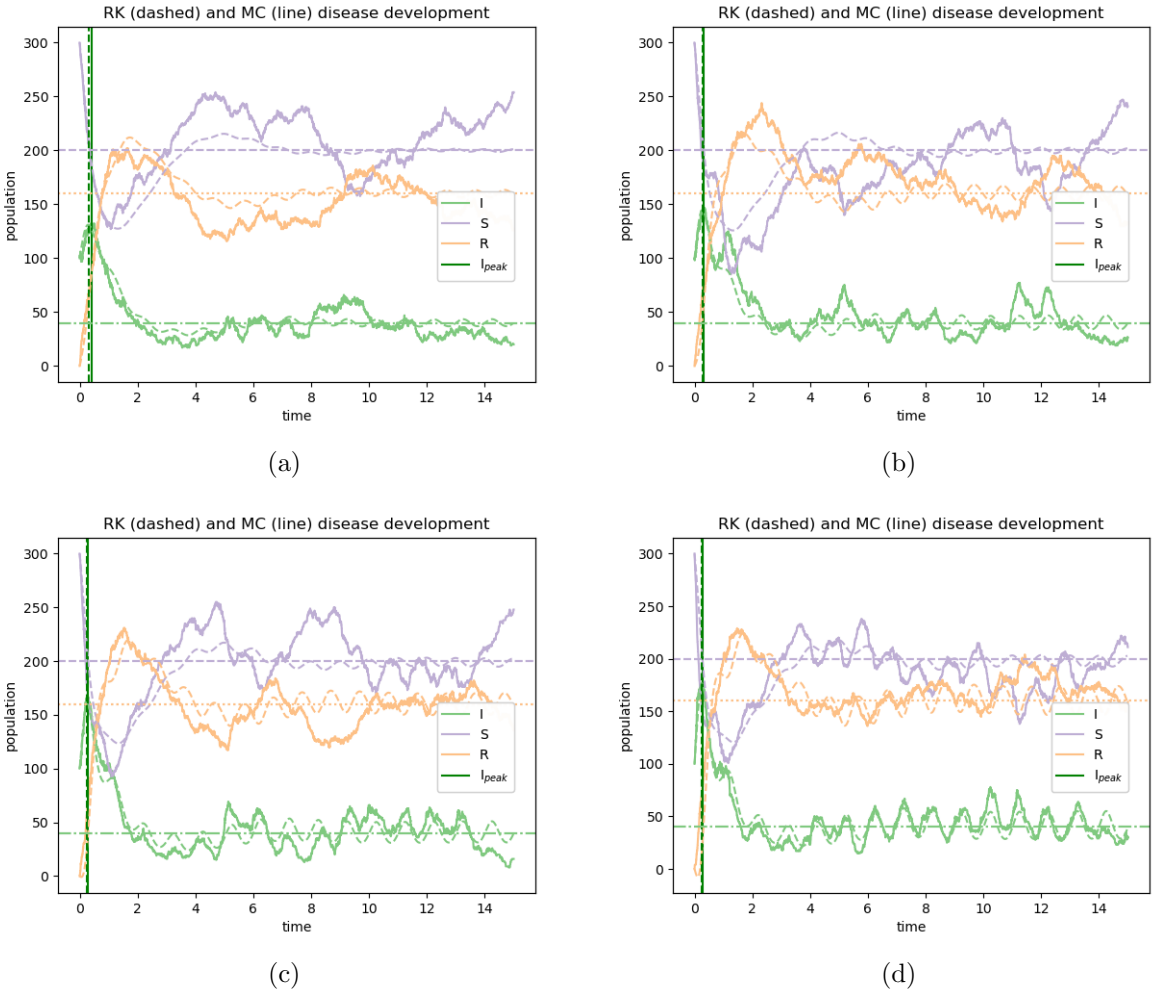
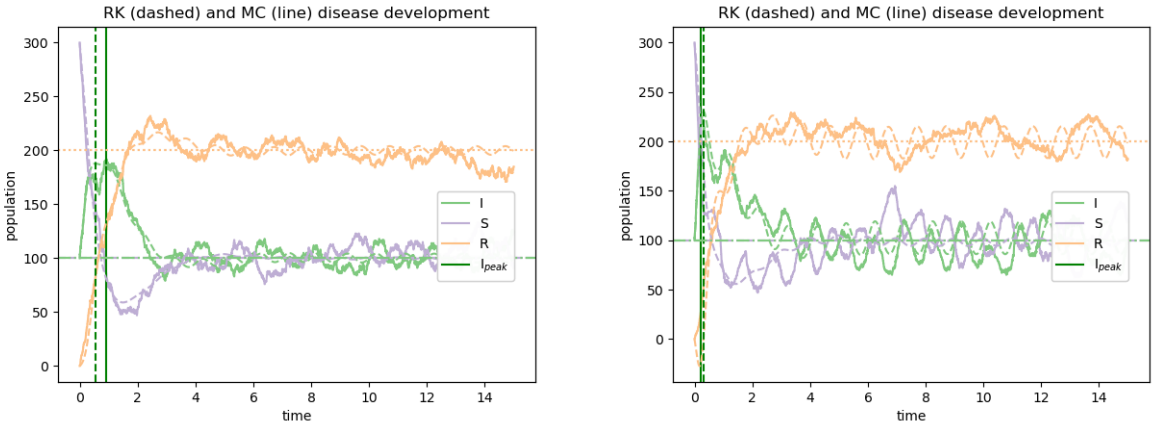
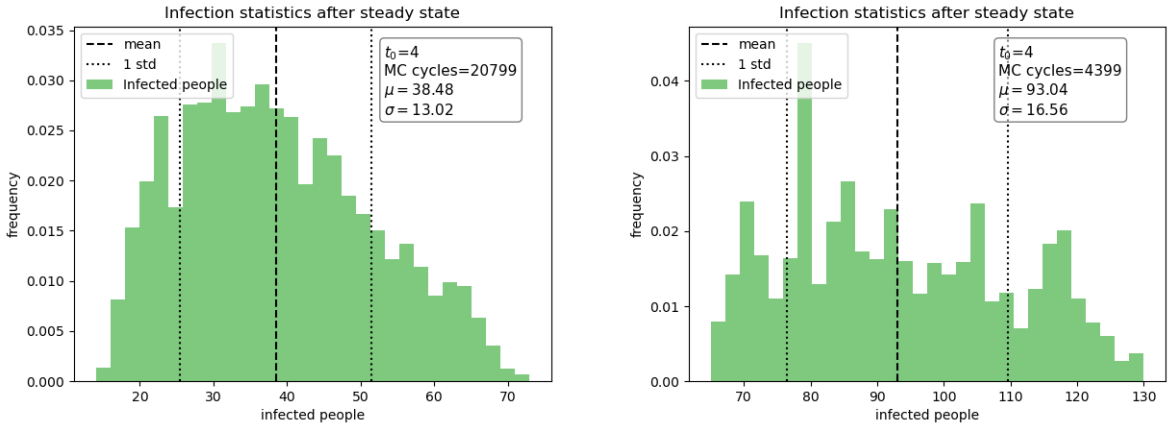


Figure 8: Disease development for $a = 4$, $b = 2$, $c = 0.5$, with seasonal variation amplitude $A \in \{1, 2, 3, 4\}$.



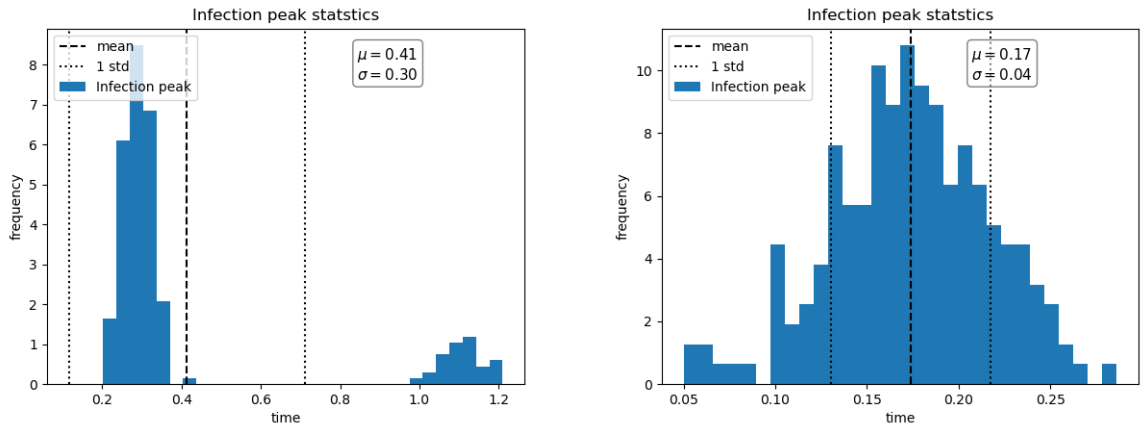
(a) (b)

Figure 9: Disease development for $a = 4$, $b = 1$, $c = 0.5$, with seasonal variation amplitude $A \in \{1, 4\}$.



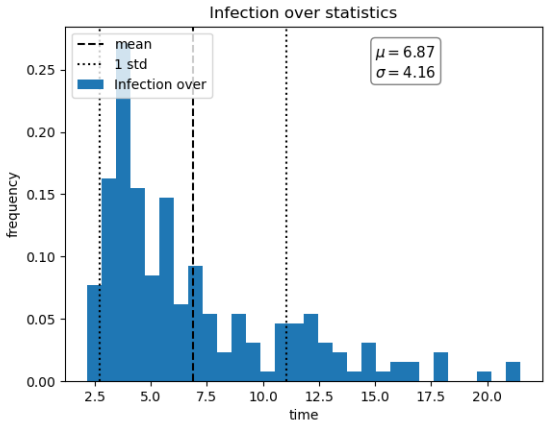
(a) (b)

Figure 10: Disease statistics for Infections after steady state for $a = 4$, $b = 1$ (a) and $b = 2$ (b), $c = 0.5$, with seasonal variation amplitude $A = 4$.



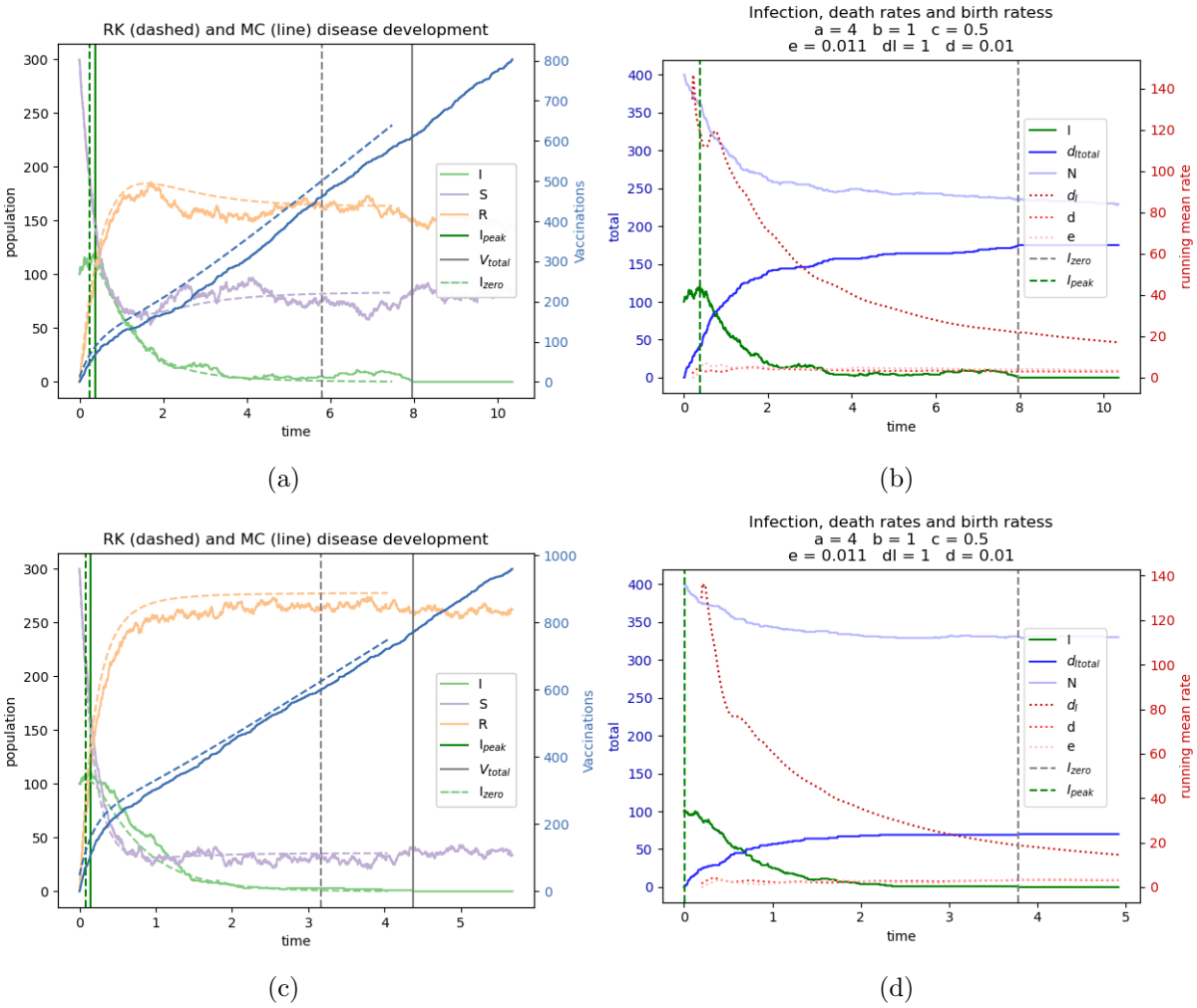
(a) (b)

Figure 11: Infection peak for $a = 4$, $b = 1$ (a) and $b = 2$ (b), $c = 0.5$, with seasonal variation amplitude $A = 4$, and vital parameters $e = 0.011$, $d = 0.01$ and $d_I = 1$ collecting statistics over 200 runs.



(a)

Figure 12: Infection over for $a = 4$, $b = 2$ (b), $c = 0.5$, with seasonal variation amplitude $A = 4$, and vital parameters $e = 0.011$, $d = 0.01$ and $d_I = 1$ collecting statistics over 200 runs.



(a) (b)

(c) (d)

Figure 13: Disease development for $a = 4$, $b = 1$, $c = 0.5$, with vaccination rates $f = 1$ (a and b) and $f = 8$ (c and d)

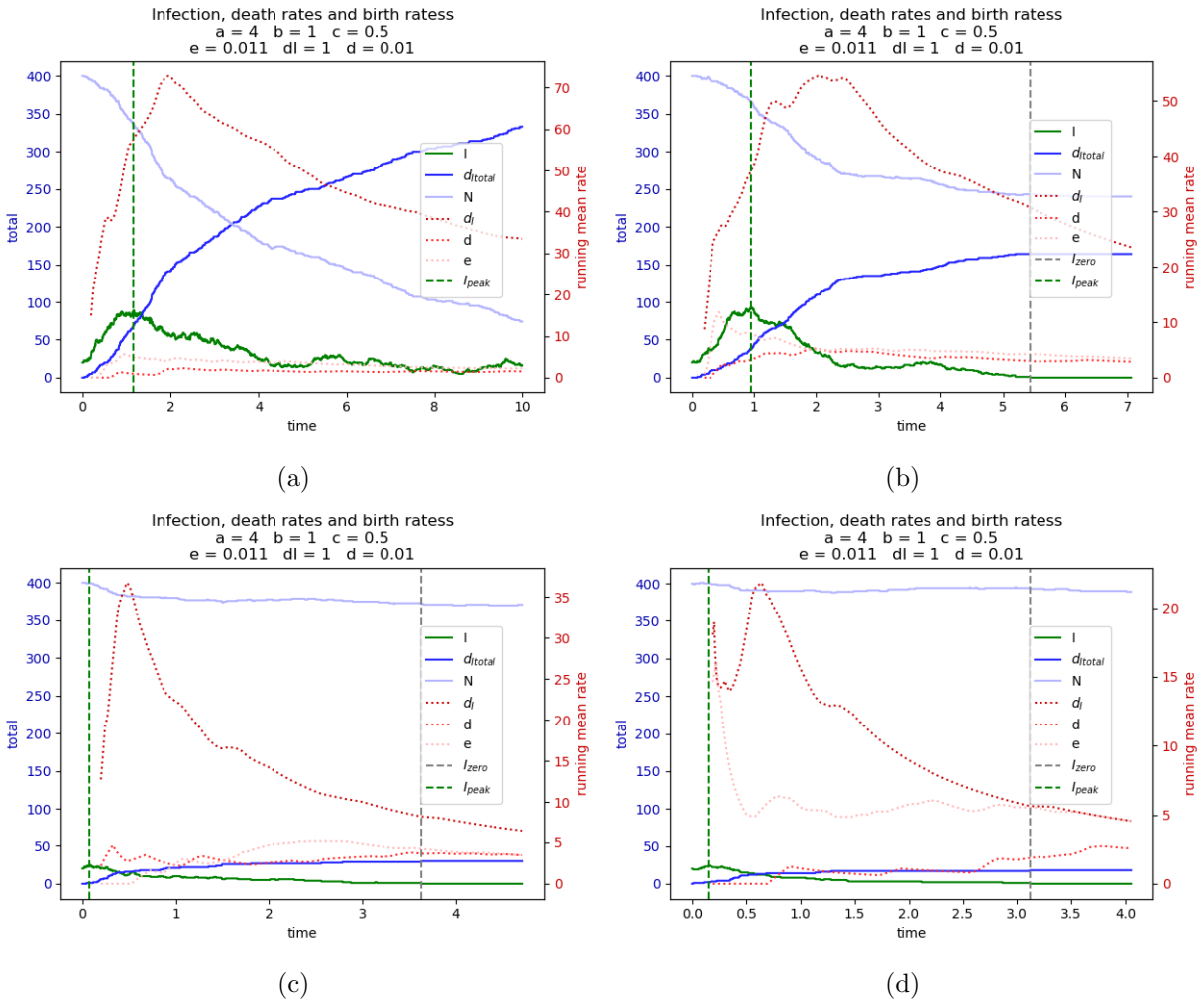


Figure 14: Vaccination rates effect on disease development for $a = 4$, $b = 1$, $c = 0.5$, with vaccination rates $f = 0$ (a), $f = 0.5$ (b), $f = 1$ (a), $f = 4$ (a)