Monte Carlo simulation and 4th order Runge-Kutta solver for the SIRS disease model

Gaute Holen — Github Repo

December 2020

Abstract

During december lockdown 2020, disease modeling is as relevant as ever, and to gain more insight into how disease behaves in an isolated population numerical models are developed. A fourth order Runge-Kutta model, and a Monte Carlo simuation, both based on the same SIRS model are used to explore vital pramaters, seasonal varitains and vaccinatins effets on disease development. The RK model is faster, while the MC simulation produces a greater variety of results which is considered closer to the unpredictable and random behaviour of a random system such as a disease in a population. Infection death rate seems to cause previuosly stable populations of indected people in the population to disappear faster, seasonal variations cause more varied outcomes and scenarios, while vaccines prove very efficient at reducing deaths and infections, particularly if introduced early.

1 Introduction

Modelling disease is as relevant as ever as COVID-19 is having a signifficant impact globally. While efforts are being made to reduce the spread and infection, the fear of the unknown, or of what we don't understand, brings out good, bad and at times very questionable protocols and policies to try to gain control over the pandemic. Disease behaviour seems to not be very well understood by most people, and saving face, showing a willingness to act, scapegoating certain groups with no voice and keeping the industry or economy alive (to some few or the many's benefit), seems to get in the way of policies based on hard science, models, simulations and expert advice from researchers. With this emotionally loaded motivation, largely based on general contempt for how horrible this semester has been, an escape from reality through a deep dive into disease modeling to try to scientifically understand why everything sucks becomes the perfect coping mechanism for a nerd not in touch with his feelings.

To model disease, the SIRS model is used with the addition of vital parameters, seasonal parameters and vaccines. Monte Carlo simulations and a fourth order Runge-Kutta solver developed in c++, with plotting methods in python using matplotlib. With a theoretical background with expected values for the simplest case, the numerical methods are validated, tested and benchmarked. Through plots, insight is gained into how disease dvelops, with vital parameters, seasonal parameters and vaccination. Lastly, the plots are discussed, and the two methods are compared. Lastly, the main findings are summarised in the conclusion.

2 The SIRS model

In short, the SIRS model is a model on an isolated population N, where each member of the population and be any of the three states:

1. S Susceptible to the disease
2. I Infected with the disease
3. R Recovering from the disease with some temporary immunity

The relationship between the different states are described by the following coupled differential equations

$$S' = cR - \frac{aSI}{N} \tag{1}$$

$$I' = \frac{aSI}{N} - bI \tag{2}$$

$$R' = bI - cR \tag{3}$$

Where a is the infection rate, b is the recovery rate and c is the loss of immunity rate. If N not time dependent,

$$R = N - S - I$$

So that

$$S' = c(N - S - I) - \frac{aSI}{N}$$

$$I' = \frac{aSI}{N} - bI$$

Otherwise

$$N = I + R + S$$

Needing I, R and S to calculate N, so then we need 3 coupled equations as S depend on N, and N depends on S when N is time dependent. By introducing birthrate e, natural deathrate d and deathrate due to disease d_I, and assuming all newborns are susceptible and that everyone can randomly die from natural causes, we obtain

$$S' = cR - \frac{aSI}{N} + eN - dS \tag{4}$$

$$I' = \frac{aSI}{N} - bI - dI - d_I I \tag{5}$$

$$R' = bI - cR - dR \tag{6}$$

Seasonal variations are introduced by modyfying the infection rate a' (also referred to later as $a_{seasonal}$), such that

$$a' = a + Acos(\omega t) \tag{7}$$

Vaccinations are introduced with a vaccination rate f, causing a direct transition from S to R, such that

$$S' = cR - \tfrac{aSI}{N} - f \tag{8}$$

$$I' = \tfrac{aSI}{N} - bI \tag{9}$$

$$R' = bI - cR + f \tag{10}$$

Transitions between each state, are described by transiton probabilites

$$P(S \to I) \tag{11}$$
$$P(I \to R) \tag{12}$$
$$P(R \to I) \tag{13}$$

Which may depend on which paramaters one might want to implement in a model. The specific implementations will also vary depending on what method is used for solving or simulating the system.

If only a, b and c are considered, the expected values for the steady state of the system is given as, which is used to verify the simplest runs of the model

$$s^* = \frac{b}{a} \tag{14}$$
$$i^* = \frac{1-\frac{b}{a}}{1+\frac{b}{c}} \tag{15}$$
$$r^* = \frac{b}{c}\frac{1-\frac{b}{a}}{1+\frac{b}{c}} \tag{16}$$

3 Algorithms, implementation, testing and methodology

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 [1]

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

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.

One thing to note, is that this specific implementation, using the transition probabilites in *Table* 1, runs through every single member of the population, and the majority of the transitions are not depending on macroscopic values. Perhaps there are ways to implement a more macroscopic MC method, which would do less evaluations and if-statements, speeding up the simulation significantly,

[1] The implementation was assisted and inspired by this blog

Transition	Transition Probability	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

however this is not done here. The intent is to make each member of the population as independent of the others as possible, and have the ability to update N and I between each persons move, so if one person gives birth, then the immidiate evaluation takes $N = N + 1$ into account, rather than just updating these things once per cycle. There is possibly lost some performance to this, however on the spectrum of "very MC" to "MC like", this implementation remains "very MC", which was the intent.

3.3 Visual methods

To make any sense of the data produced by the algorithms, three major plotting methods are implemented:

1. SIR plot, showing the S, I, R and the total number of vaccines given as a function of time for the Monte Carlo simulation (full line) and the Runge-Kutta solver (dashed line). it also has horizontal dashed lines indicating the expected values when N is constant (where the expected values are accurate). There are vertical lines (full line MC, dashed line Rk) showing infection peaks, and when the infection is over, which is defined as less than one person having the infection.

2. Id_I plot, shwoing the number of infected people, the population N, and the total number of deaths due to the disease on the left y-axis as a function of time. On the right y-axis, it shows the running mean rates as dotted lines for e, d and d_I. This has to be a running mean, as for the MC simulation, the deaths and births only happens once per timestep, so the plot would just be a bunch of discrete spikes going between 0 and 1 at different frequencies. By applying a quite heavy running mean filter, a smooth curve is prodcued, however this curve is shifted in time. The time shift is not constant, and to avoid assuming misrepresenting the curve as perfectly asjusted to account for timeshift, this artifact is left there to give some indication as to how heavy the plot is processed. In addition, as in the SIR plot, there are vertical lines indicating infection peak and when there are less than one person infected.

3. $\mu\sigma$ plot, showing the standard deviation, mean and distribution of input data. This is used wither to find mean μ and σ after the steady state is reached (green), or to find the range for infection peaks and infected people being less than 1 for consecutive runs of the MC simulation (blue). This method aims to show how much variation in the outcomes there is for the MC simulation.

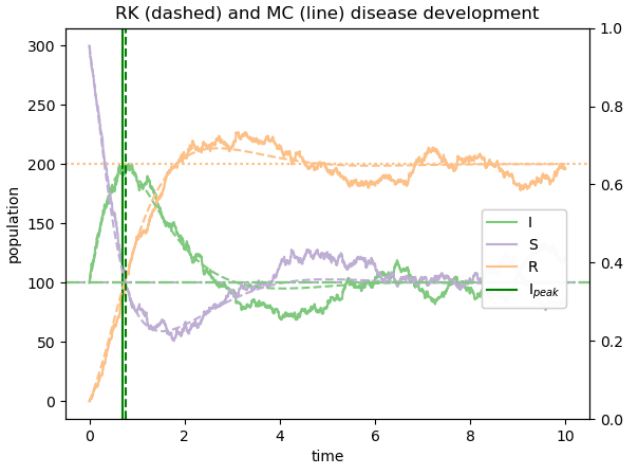


Figure 1: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with $a = 4$, $b = 1$ and $c = 0.5$

3.4 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.

3.4.1 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.

3.4.2 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.

3.5 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.

3.5.1 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

$$N = I + R + S \tag{17}$$

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.

3.5.2 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.

3.5.3 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

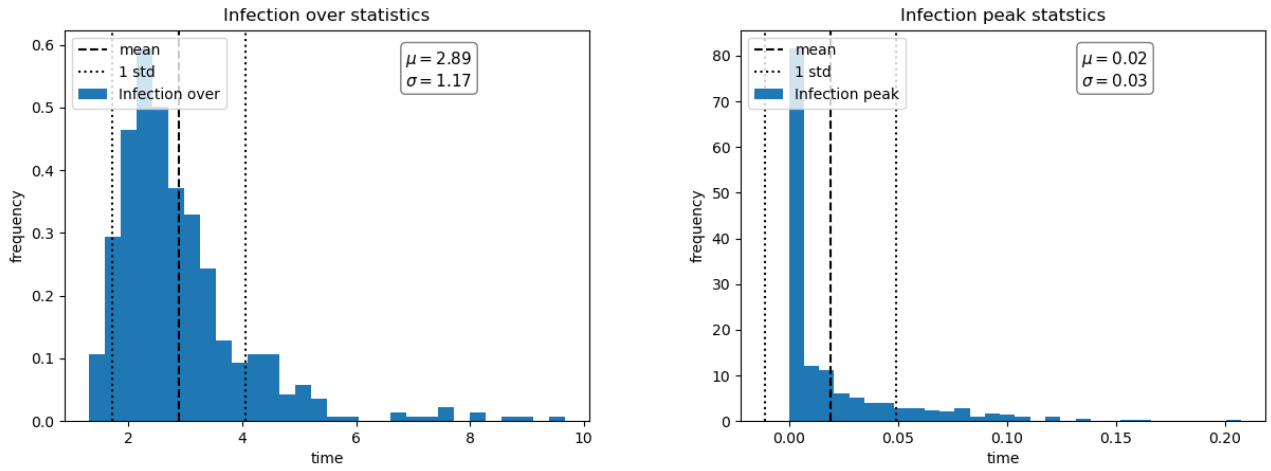
Table 3: Runtimes with different compiler flag opimizations for single run RK and multiple run MC.

4 results

Here, results, in the form of the visual methods described in the previous section, are presented. Only descriptions of the plots, and why certain values were picked is discussed here. For thorough discussion and analysis of the results, see the section for discussion and analysis. In order to make results somewhat comparable, an attempt to keep the parameters the same as much as possible is made.

4.1 Error and accuracy

The first results presented are some statistics for the simplest case with only a, b and c considered.



(a) (b)

Figure 2: The time taken for the when the infection is over (a), and when the ifection peaks (b) $a = 4$, $b = 4$ and $c = 0.5$ run 500 times each

In *Figure* 2, 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.

In *Figure* 3, the mean value of the number of infections after reaching a steady state is calculated.

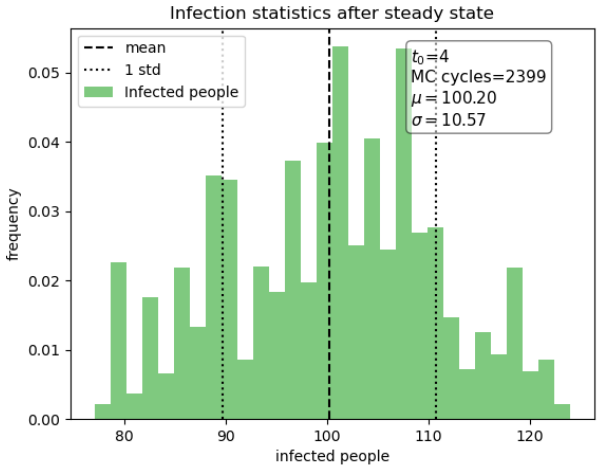


Figure 3: MC statistics after the steady state with $a = 4$, $b = 1$ and $c = 0.5$ for 500 runs

4.2 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 $b \in \{1, 2, 3, 4\}$.

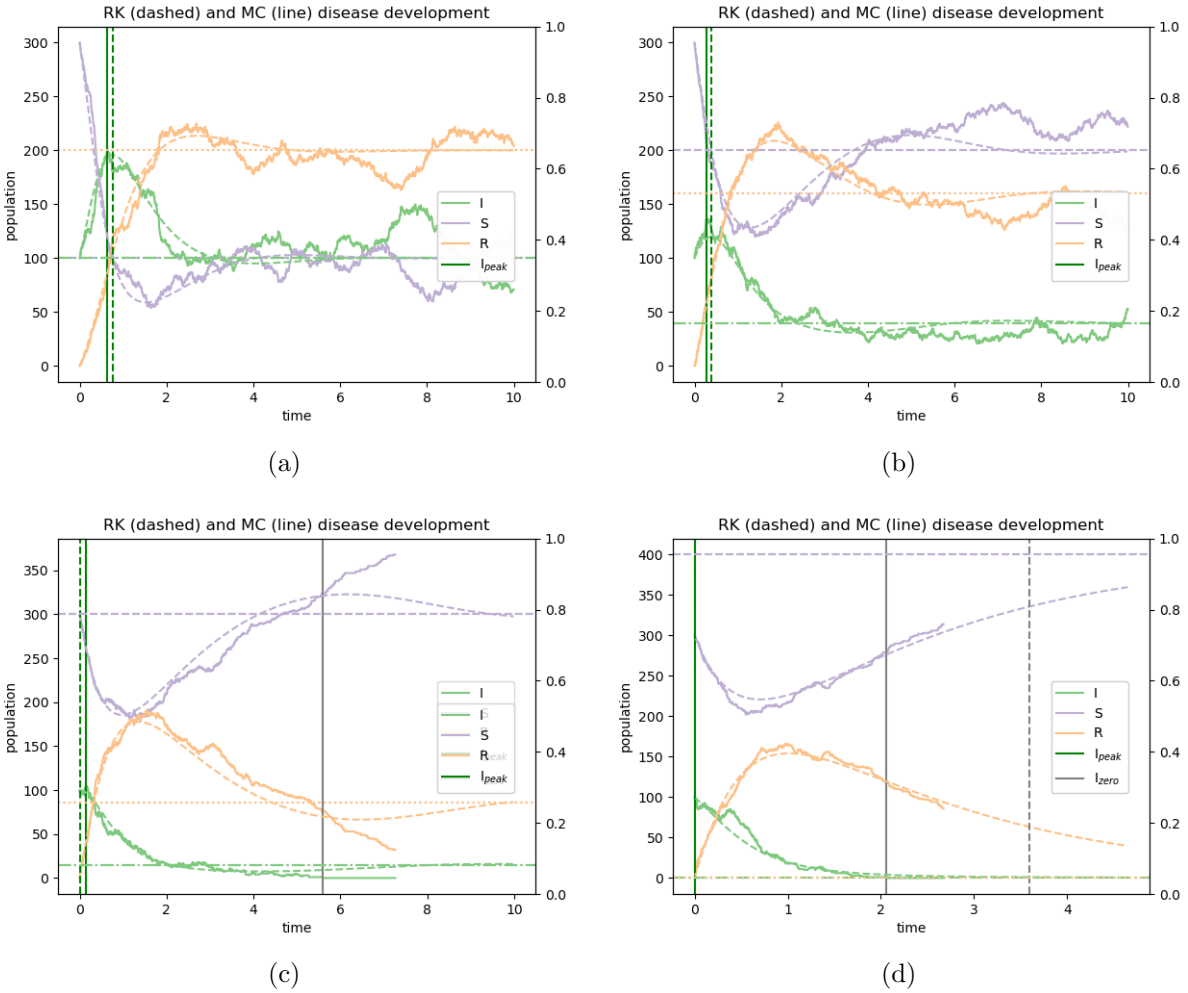


Figure 4: Comparison of RK solver (dashed) and MC simulation (line) with expected values (horizontal dashed) with fixed $a = 4$ and $c = 0.5$. b has values 1(a), 2(b), 3(c) and 4(d). The horizontal dashed lines are the expected values.

In *Figure* 4, 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 $b = 2$.

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. Having the birthrate slightly higher than the natural death rate, makes the infection deaths decide whether the population is increasing or decreasing, which makes interpretation of the plots easier. Some key plots will be included to show how d_I affects the steady state, and how fast the infection is over.

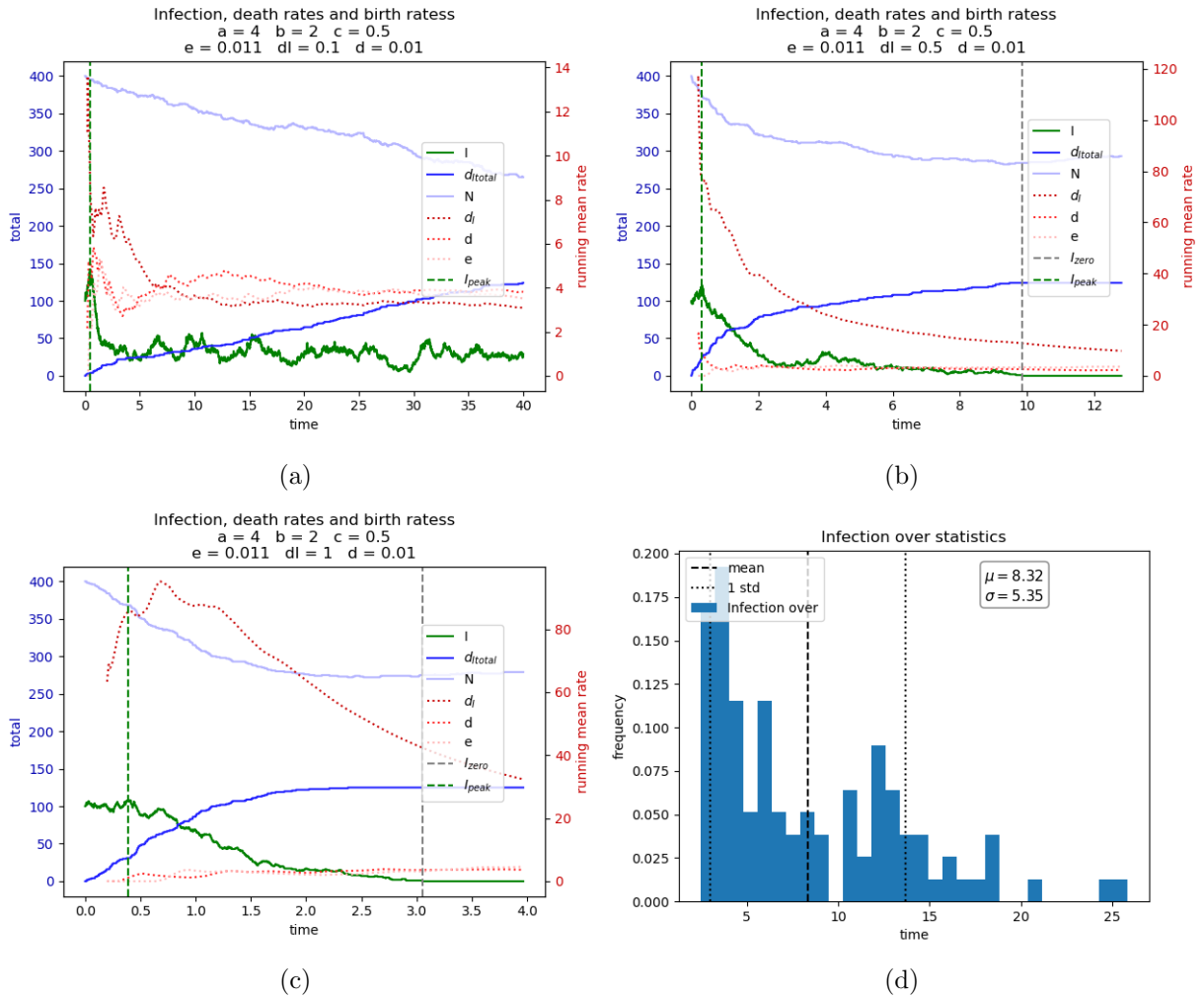


Figure 5: Disease development for $a = 4$, $b = 2$, $c = 0.5$, $e = 0.011$ and $d = 0.01$. d_I varies as 0.1 (a), 0.5(b) and 1(c and d), with 100 runs collecting the statistics in (d)

In *Figure* 5, varying values of d_I are shown for the previously stable case where $b = 2$.

In the case of the MC solver, having d_I high may cause the disease to die faster due to randomness. 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* 6 the initial infections are set to 20, but the disease development is still the same, just with a lower peak than before. Here, it is hard to tell whether the total deaths are lower or higher, since when we start with 100 indected, leading up getting 100 infected, a lot of people probably died. So, the most reasonable thing to look at is the height of the infection peak, and the death rate, which is lower in the case with initial $I = 20$, meaning that a higher deathrate probably reduces the infection peak deaths by killing carriers.

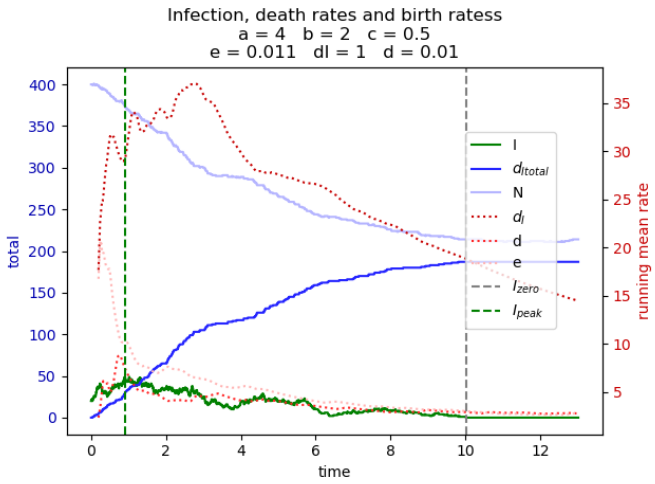


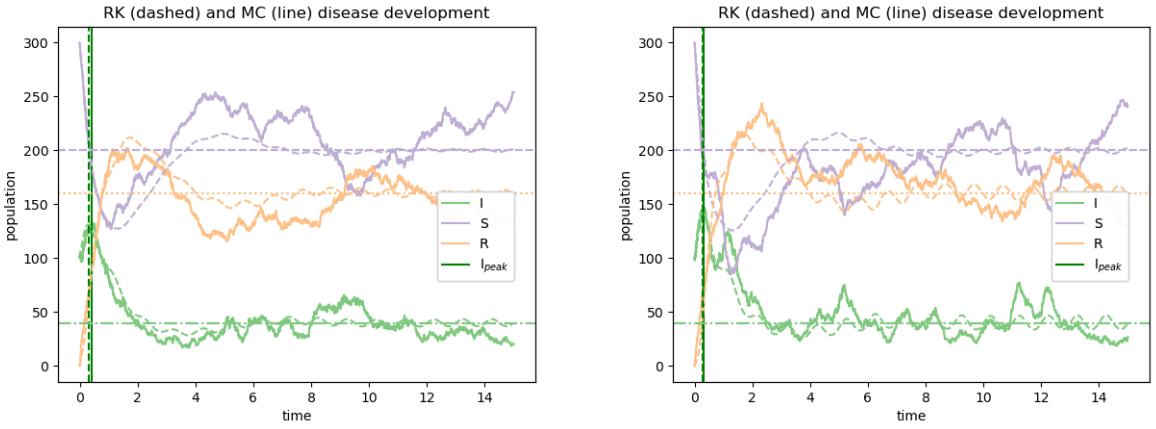
Figure 6: A run with initial infections = 20, with $a = 4$, $b = 2$, $c = 0.5$, $e = 0.011$, $d = 0.01$ and $d_I = 1$.

4.4 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

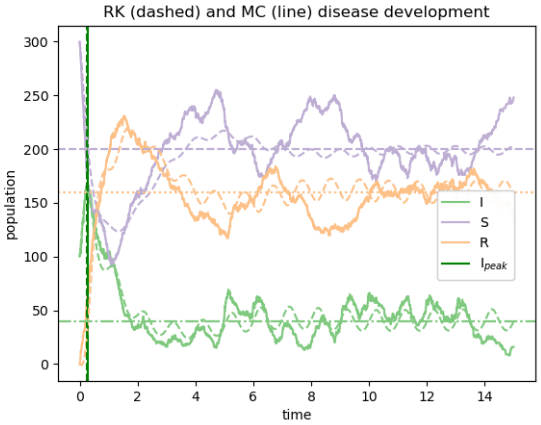
$$a_{seasonal} = a + Acos(\omega t)$$

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.

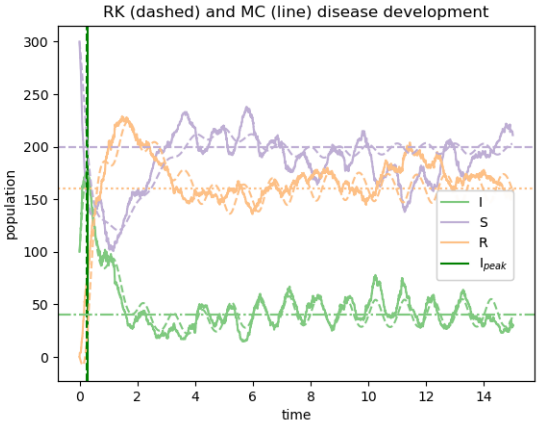


(a)

(b)



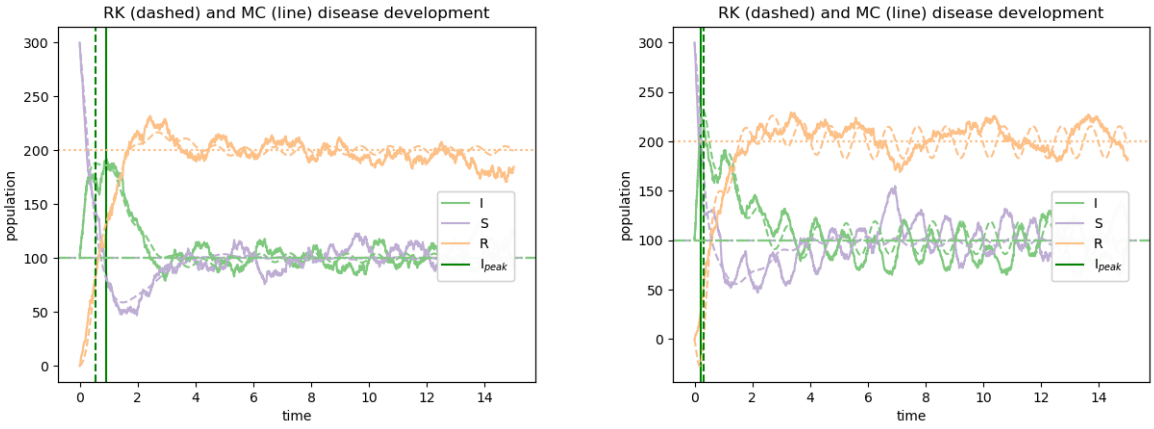
(c)



(d)

Figure 7: Disease development for $a = 4$, $b = 2$, $c = 0.5$, with seasonal variation amplitude $A \in \{1, 2, 3, 4\}$.

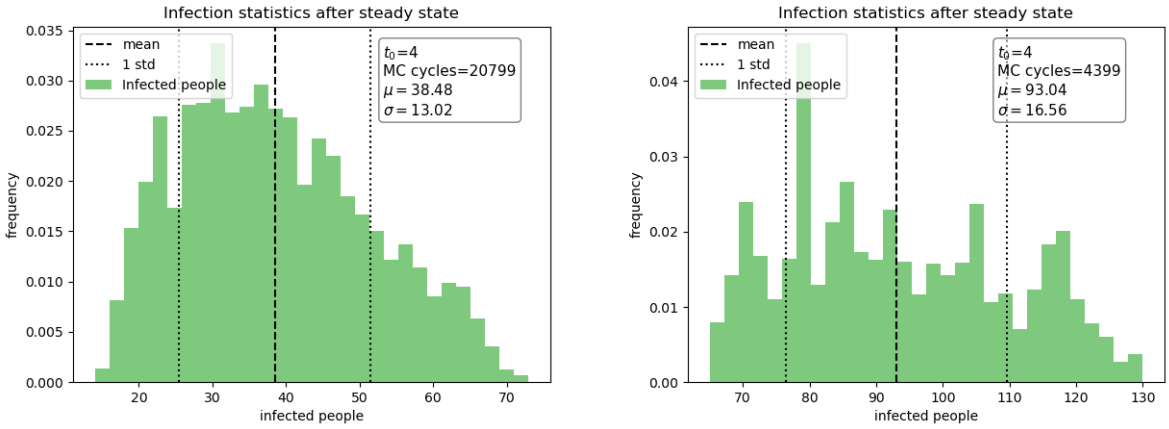
Changing b so that $b = 1$ causes the expected values for I and S to be the same, highlighting the cyclic relationship of the two, as seen in In *Figure* 8



(a) (b)

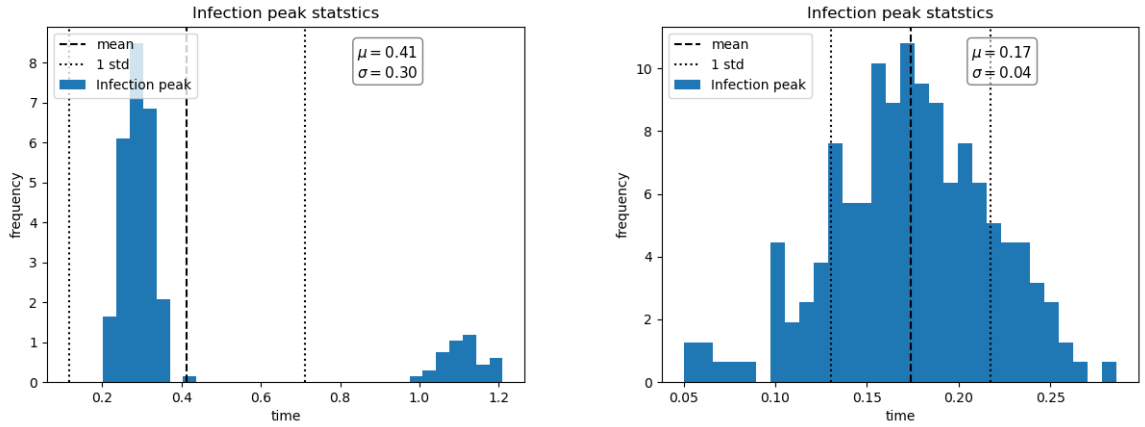
Figure 8: Disease development for $a = 4$, $b = 1$, $c = 0.5$, with seasonal variation amplitude $A \in \{1, 4\}$.

So how does this affect the statistics of when an infection is over, or the mean amount of infected people? In *Figure* 9 the mean remains the same as in *Figure* 3, but σ is higher.



(a) (b)

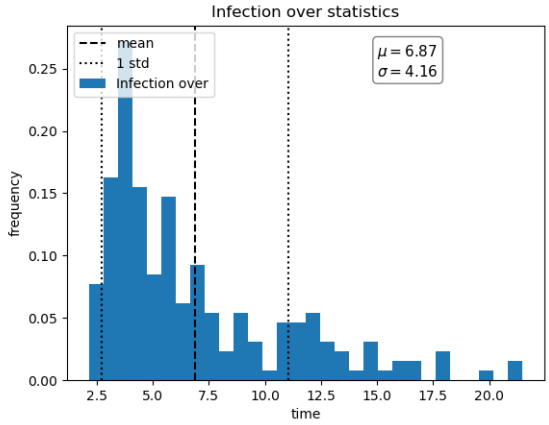
Figure 9: Disease statistics for Infections after steady state for $a = 4$, $b = 1$ (a) and $b = 2$ (b), $c = 0.5$, with seasonal variation amplitude $A = 4$.



(a) (b)

Figure 10: 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.

s In *Figure* 10 (a), there is actually a chance of having a delayed peak on the second cycle, rather than the first. This however is not the case in (b). In *Figure* 10, compared to *Figure* 5, the σ values are about the same, so no here the seasonal variation's effect is inconclusive.



(a)

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

4.5 Vaccines

The next scenario to be considered is vaccines. The case studied here in particular, is how vaccines can prevent deaths, and flatten and reduce the peak of the infection curve. In *Figure* 12, the simulation is stared at $I = 100$, meaning that many already are infected, so here vaccination will cause the amount of infected people do drop faster, due to fewer being susceptible. Is the inital conditions are close to the peak, the actual peak is mostly unaffected, with the disease duration

being the main thing that is changed. However, in (b) and (d), the total deaths are much lower in (d), with a similar amount of total vaccines given. Thus, vaccinating many fast is better than many slow.

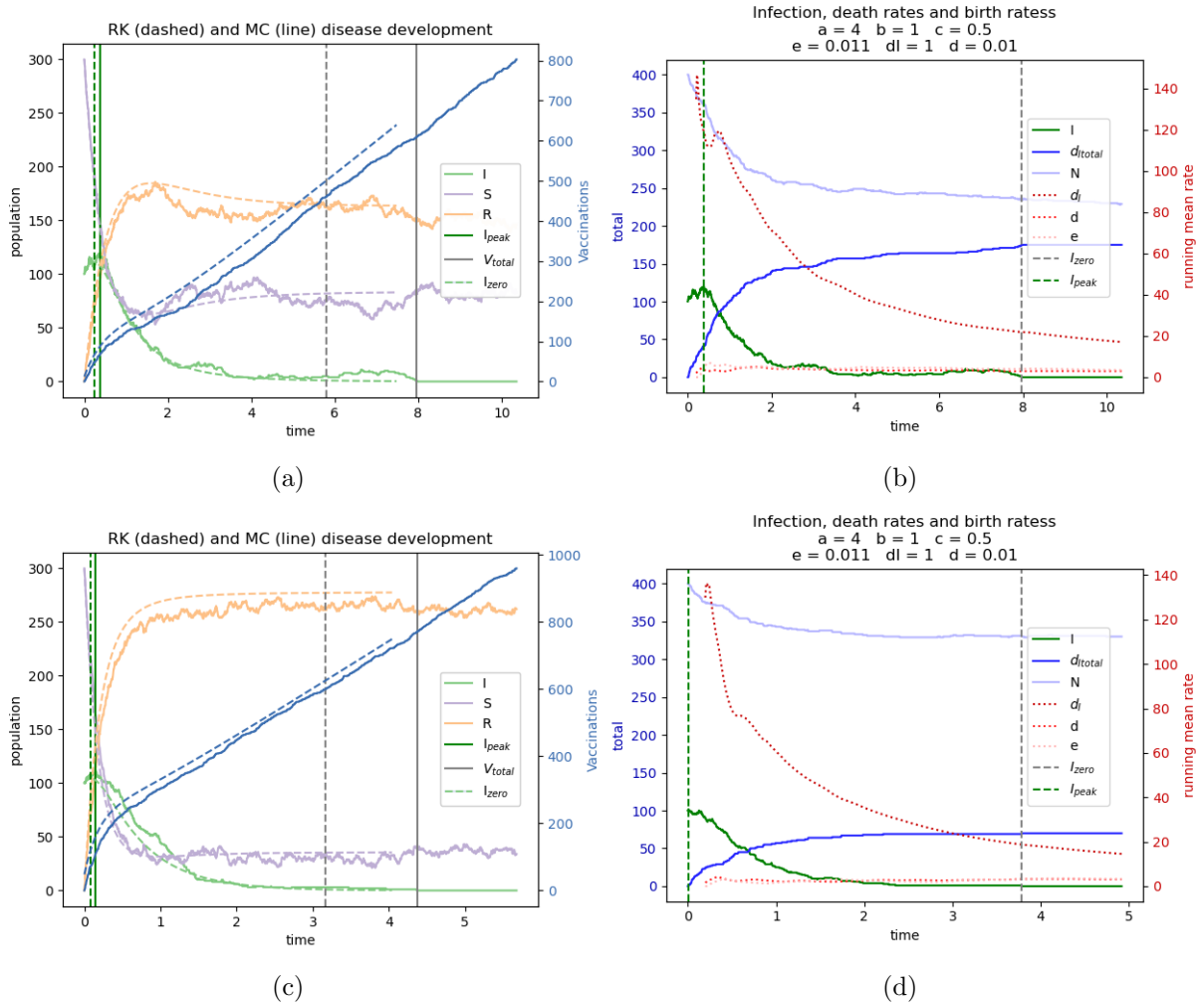


Figure 12: Disease development for $a = 4$, $b = 1$, $c = 0.5$, with vaccination rates $f = 1$ (a and b) and $f = 8$ (c and d)

In *Figure* 13, vaccines are introduced early, starting the simulation at $I = 20$. From *Figure* 6, we saw that starting the simulate early at $I = 20$, produces the same kind of peak than starting later, given that there are no factors such as very high disease death rate significantly halting the disease on its way to its pieak. Vaccination can be considered one suck peak reducing factor, as seen in In *Figure* 13, where early vaccination of just $f = 0.5$ drastically reduces total deaths. For $f = 1$ and $f = 2$ in (c) and (d), this effect is even more dramatic. So, introducing vaccines early can have a very severe impact on the total deaths due to the disease.

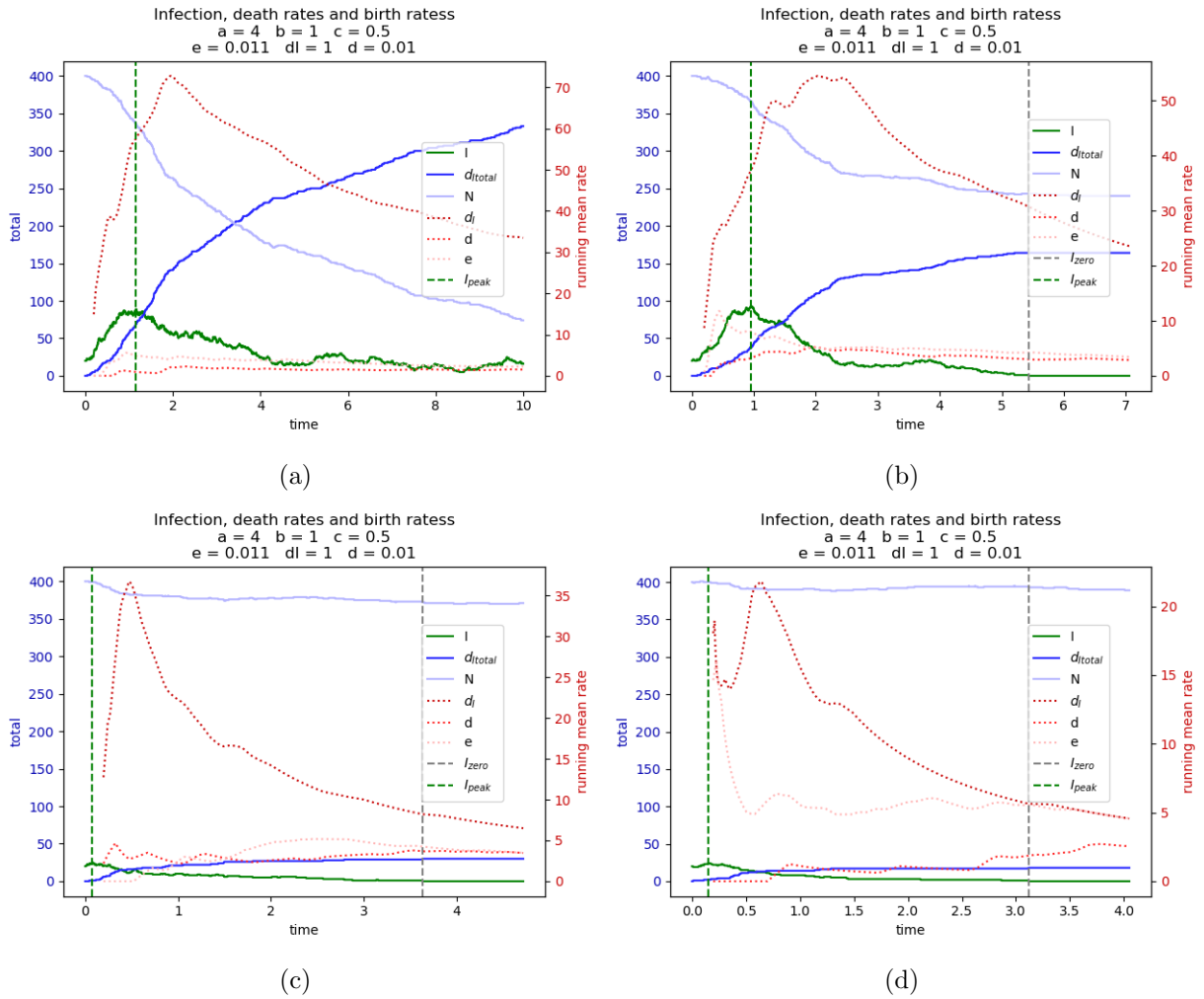


Figure 13: Vaccination rates effect on disease development for $a = 4$, $b = 1$, $c = 0.5$, with vaccination rates $f = 0$ (a), $f = 0.5$ (b), $f = 1$ (c), $f = 4$ (d)

5 Discussion and Analysis

One thing to note, is that the data shortly after the infection dies out is cut for the MC solver and RK solver. This is because shortly 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 and gain some insight from it, not to map every single combination of inputs possible.

5.1 Accuracy of models

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.

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.

After reaching the steady state unequal to zero, statistics are also collected for the amount of infected people, 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 with the steady state statistics.

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. Although this might seem like a weakness, it also gives an indication as to how small random cases of infection can have a butterfly effect and severly affect when the peak is, how high it is and how long the disease stays in the population. The RK solver gives an exact solution for this, but doesn't really have any way to extract data on the variety of outcomes given one set of initial conditions. As disease spread is a random process, the ability to compare σ values to determine the scope, or domain, of possible outcomes, can be at least as valuable as an exact solution.

5.2 Insight into disease modeling

5.2.1 Steady state with infection in populaiton

From the equations for expected values, we need $b < a$ in order for an infection to reach a steady state within a population.

However, due to the randomness in the MC solver, as discussed above, in *Figure* 4 (c) 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. The interesting takeaway is that the MC solver can produce outcomes where the disease dies out when it has a dip which is expected to increase again later, something the RK solver can't do.

5.2.2 Vital parameters effect on extinction of disease

For the vital parameters, the RK solver plots were left out, as they converged to zero if

$$e - (d + \frac{d_I I}{N}) < 1$$

and diverged otherwise.In other words, the vital parameters only acted the same as *Figure* 4 multiplied with a convergent/divergent constant with very predictable behaviour. For this reason, only the MC runs are discussed here, as they produced more interesting results.

The main thing to look for with the death rates, is whether the previously stable cases with $b = 1$ and $b = 2$ will become unstable and converge to zero infections if enough infected people die. When the carriers of the disease die, less people are infecting others, causing the disease to be less infectious, making the $b < a$ criteria for infections to stay in a population no valid, as a is in practce lowered by infected people dying before infecting others.

By starting the simulation earlier, the impact of the disease death rate, while keeping it reasonably low, d_I does not self handicap to the extent where it prevents the disease from even existing. Obviously, if the deathrate is unreasonably much higher than the infection rate, then the first few people getting the disease would just die before it could spread. As a result, the deathrate's effect is not explored for unreasonable extremes. That being said, the infection peak, and total amount of deaths was not really affected by increasing values of d_I in *Figure* 5. The only thing that was affected, was the time scale where the deaths occured, which decreaed with increading d_I.

5.2.3 Seasonal parameters effect on pandemic duration and peak

With seasonal variations, 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? According to *Figure* 7 and *Figure* 8, I cycles about the expected values both for MC and RK. Something that was shown to be impacted with severe seasonal variations, is that the infection peak time could be delayed one cycle in certain cases. This could have significant impact as one might think a pandemic has passed its peak as infections are going down, before it changes and has another even higher peak.

The σ values were higher for steady states, and in general the seasonal variability adds unpredictability to the model, increasing the domain of outcomes and variations.

5.2.4 The effects and timing of vaccination

As for vaccines, they have the most severe impact on reducing infections and deaths. An early vaccination seems to almost completely negate the infections, and cause the disease to almost disappear. If there was ever any doubt that vaccines work, this is another indcation that they indeed do work very well.

5.3 MC vs RK, strengths and weaknesses

Now, that the results have been discussed, and benchmarks and runtimes have been asessed, one can make a comparison between the MC and RK methods. When simulating a random system, such as disease development, which can have a great variety of outcoms for the same initial conditions, the MC solver, particularly with the statistcal methods with σ values, is preferred over the RK solver which produces the same answer every time. That being said, the RK solver is significantly faster, and if the population were to be increased, then the MC solver would be much slower. As a result, if there are some specific population size that is hard to simulate with MC, a compromise where a downscaled MC simulation with the RK solver could be used instead. If very fast runtimes are a concern for simple scenarios, then the RK solver is maybe better. But, for more complex schenrios, the MC solver is preferred due to its ability to produce a range of outcomes, and give an indication as to what range of outcomes one could expect.

6 Conclusion

To conclude, for the simpler cases where the thoretical expected values are valid, both the MC and RK solver produce similar results. The only advantage the RK model is its speed, as the

MC model produce a variety of outcomes, and can define a domain of possible outcomes rather than just the most likely one, giving insight to possible, less likely scenarios that are still worth considering. Through the many runs and results presented, increasing d_I seem to make infections less likely to reach a non-zero steady state in the population. Additionally, seasonal variations add more variety to the onset and peak of the infection, while the time where the infection ends seems to be unchanged. Lastly, vaccines are an extremely powerful tool, where if they are introduced early they can almost completely negate a pandemic with very few deaths.

References

[1] "Blog with an example of RK4 method" Accessed December 2020 from https://blog.tonytsai.name/blog/2014-11-24-rk4-method-for-solving-sir-model/