Simulating the Spread of Disease using the SIR model with both an Iterative and Monte Carlo approach.

Caspar William Bruenech

18. desember 2019

Sammendrag

Two different methods for simulating the spread of disease using the SIRS model is presented. A Monte Carlo approach allowed the introduction of more realistic discrete transitions between the different groups, but required more CPU time to produce accurate results, while the Runge Kutta method proved to be superior in terms of speed, accuracy, and its ability to visualize how the populations change over time, something the Monte Carlo implementation was unable to do. Nevertheless, the two methods provide interesting and different approaches to solving coupled differential equations, and can both be used depending on the experimental situation.

1 Introduction

The SIR model is a compartmental model technique for modeling the spread of infectious disease mathematically [1]. By including various parameters, these models can be used to predict how a disease will spread through a population, what will be the outcome of an epidemic, or how a vaccine should be issued in the most efficient way. In this experiment we will begin by introducing the simple SIR model in the form of coupled differential equations, before adding various complexities to the model, including vaccination and seasonal variation. The equations will then be simulated using both the iterative Runge Kutta method, and with a Monte Carlo approach by introducing discrete transitions between the different population groups. By studying four different populations which are differentiated by their rate of recovery, we will study how the addition and variation of the different parameters affect how the populations develop over time. Finally we will look at the up- and downsides of the two methods for solving the equations, and discuss whether one is better than the other.

2 Method

In this experiment we will study the spread of disease by implementing the already-mentioned SIR model, named after the three main groups of people defined by the model:

1. Susceptible (S) - those without immunity, who can be infected,
2. Infected (I) - those who are infected with the disease,

3. Recovered (R) - those who have been infected but are now disease free and have developed immunity to the disease.

The model defines the possible transitions from one group to another only as $S \to I \to R \to S$. Additionally, the parameters a, b c define, respectively, the rate of transmission, rate of recovery, and rate of immunity loss. In our implementation of this model we will also assume a homogeneously mixed population, and that the total number of people in the population N is constant, i.e,

$$N = S(t) + I(t) + R(t) = constant.$$

For all variations of this model to be studied in this article, we will be analysing four different populations with the initial population counts

$$S(0) = 300 \qquad I(0) = 100 \qquad R(0) = 0.$$

2.1 A Simple Case

Initially we will study the most simple example where we ignore the effects of birth and death rate of the population. This means that, for this model, people can only get sick or get sick and recover. With these assumptions, the coupled differential equations describing the populations can be created as [1]

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

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

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

While this system does not have analytical solutions, the equilibrium states can be obtained by utilizing the constraint of a constant population count. Using this, the problem is reduced to

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

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

By setting these equations to zero, the fraction of people in each group at equilibrium are given as

$$\frac{S^*}{N} = \frac{b}{a}$$

$$\frac{I^*}{N} = \frac{1 - b/a}{1 + b/c}$$

$$\frac{R^*}{N} = \frac{b(1-b/a)}{c(1+b/c)}.$$

For the four populations to study with this model, we will differentiate them by varying the value of the rate of recovery, b, such that we have the populations A, B, C and D with the respective b-values $1, 2, 3, 4$.

By initially treating the population as a continuous variable, we can solve the coupled differential equations using the fourth-order Runge Kutta method as described in appendix A.

Realistically, however, the populations are discrete variables. To account for this, we can solve the system using Monte Carlo methods by introducing transition probabilities such that the populations are incremented discretely. From equations 1, 2, 3, we observe that the amount of people going from S to I is equal to the negative term in S' and the positive term in I'. The same is true for populations going from I to R and R to S, giving us the approximated values for the number of people going between groups in a time step Δt as

$$S \to I \approx \frac{aSI}{N}\Delta t, \quad I \to R \approx bI\Delta t, \quad R \to S \approx cR\Delta t. \tag{4}$$

We can then constrict the time step Δt to be small enough such that no more than one person moves from one group to another. I.e. we want to set Δt such that $A \to B = 1$. Each population can have the maximum value of N (which is equal to everyone being in one group), which again means that the maximum value for the number of people moving from one group to another in a given time step is

$$max\,(S \to I) = \frac{a}{N}\frac{N^2}{2^2}\Delta t, \quad max\,(I \to R) = bN\Delta t, \quad max\,(R \to S) = cN\Delta t.$$

By then setting these equations to 1 and solving for Δt, we get three expressions for the time step. By choosing then the smallest value, we are guaranteeing that no more than one person will move from a given group to another. I.e.

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

As this construction ensures that the values in 4 are in the range $[0, 1]$, we can treat these values as transition probabilities, which gives us the possibility of solving the coupled differential equations using Monte Carlo methods. The approach is similar to the Metropolis-Hastings sampling method; for each possible move (S to I, I to R, or R to S), a random number between 0 and 1 will be generated. If the number is less than the probability computed by 4, the move is taken. This means that, for example, if the move from S to I is approved, the population count of S is decreased by 1, while the population count for I is increased by 1. By performing M Monte Carlo cycles, we can then compute the expectation values and variations for each of the population counts by taking the average of quantities over the M cycles.

2.2 Adding Complexity

To make the model more realstic and flexible, we will be adding complexity with three parameter additions; vital dynamics (birth and death), seasonal variation, and vaccination.

2.2.1 Vital Dynamics

In the previously described simple model we have assumed that the total time span of the simulation is small enough to justify ignoring the birth and death rate of the population such that the total number of people remains constant. To make the model more realistic we will now include these parameters to the equations. More specifically, we will introduce the birth rate e, the death rate as a result of natural causes d, and the death rate as a result of the illness d_I. By assuming that new born babies are instantly susceptible, we get the new differential equations

$$S' = cR - \frac{aSI}{N} - dS + eN, \tag{5}$$

$$I' = \frac{aSI}{N} - bI - dI - d_I I, \tag{6}$$

$$R' = bI - cR - dR. \tag{7}$$

As with the initial simple model, we will solve this using both the RK4-method and a Monte Carlo method. Solving using the RK4 algorithm is a trivial case of simply changing the function by adding the new parameters. The inclusion of these parameters also results in the total population N no longer remaining constant, which means that we have to update this for each time step in both the RK4 and Monte Carlo solver. When implementing this model into the Monte Carlo solver, we still have the same transition probabilities as before, as the added parameters do not move any people between groups. However, we still only want a maximum of one person to be born or die at each time step, which means we need to include this when computing the time step. The maximum values for the probabilites of birth and death are then

$$max(P_{birth}) = eN, \qquad max(P_{death}) = dN, \qquad max(P_{death,I}) = d_I N,$$

which gives us the new value for the time step as

$$\Delta t = min\left(\frac{4}{aN}, \frac{1}{bN}, \frac{1}{cN}, \frac{1}{eN}, \frac{1}{dN}, \frac{1}{d_I N}\right).$$

Since the total population count $N = S + I + R$ is no longer constant, we need to compute this time step at each Monte Carlo cycle to ensure the transition probabilities remain in the domain $[0, 1]$.

2.2.2 Seasonal Variation

To account for the seasonal variation of a disease (such as the flu season during the colder months of the year), we can alter the rate of transmission, a, such that it varies as a function of time. One way of implementing this is by way of a cosine function, such that the new rate of transmission is given by

$$a(t) = A_0 \cos(\omega t) + a_0,$$

where A_0 is the amplitude of the rate of transmission, ω is the frequency of the oscillation, and a_0 is the mean transmission rate. To implement this, we now need to make sure we update the rate of transmission for each time step.

2.2.3 Vaccination

The final complication we will add to the model is the possibility of vaccination. This allows a susceptible person to become immune to the disease without first having it, causing a person to move directly from the S group to the R group. By then introducing the parameter f as the rate of vaccination, the final model with all the added parameters is then

$$S' = cR - \frac{a(t)SI}{N} - dS + eN - fS, \tag{8}$$

$$I' = \frac{a(t)SI}{N} - bI - dI - d_I I, \tag{9}$$

$$R' = bI - cR - dR + fS. \tag{10}$$

As with earlier introduction of new parameters, we need to include the new transition probability for a person getting a vaccination or not.

2.3 Simulating the Spread of Disease

The method for simulating the SIR model is then to solve the differential equations defined by equations 1 to 3 using both the Runga Kutta method and the Monte Carlo method. We can then vary the values of the parameters and see how these variations affect the resulting populations over time. Due to the high amount of possible parameter combinations, the birth rate e, natural death rate d, and the frequency of the oscillation of the rate of transmission ω, will be fixed as constants with values

$$e = 0.02, \qquad d = 0.01, \qquad \omega = \frac{2\pi}{T/2}.$$

The parameters to vary are then the death rate from the disease, the amplitude of the rate of transmission oscillation, and the rate of vaccination.

2.4 Benchmarking

In order to compare the Monte Carlo implementation with the Runge Kutta method, we can vary the number of grid points and Monte Carlo cycles. We can then compare the CPU time taken, in addition to the error of the resulting S, I, and R-values, which can be computed using the equilibrium equations as defined in section 2.1. Since varying number of grid points in the Runge Kutta method does not necessarily scale the CPU time taken in the same way as varying the number cycles in the Monte Carlo implementation, we can compare the two methods by plotting the error divided by the CPU time as a function of the CPU time. This will give us an indication of how the error scales as a function of number of the computational cost of each method. The error will be computed as the absolute relative error

$$error = \left| \frac{X' - X}{X} \right|$$

where X' is the approximated value of X computed with the Runge Kutta or Monte Carlo method, and X is the exact value.

3 Implementation and Results

We present a selection of the results (parameter combinations). More results can be found in the attached github[1].

For all the models, we performed 40000 Monte Carlo cycles, however we varied how large part of the result to save depending on the population. This is because of the fact that since Δt depended on the parameters of the models (including b), we observed that the different populations required a different amount of cycles in order to produce a result similar to that of the Runga Kutta solver. Therefore, for populations A, B, C, and D, we display the results from, respectively, 10000, 20000, 30000 and 40000 Monte Carlo cycles.

For the simple model, all the parameters apart from a, b and c were set to zero. The result is shown in figure 1.

An example of the inclusion of vital dynamics is shown in figure 2, with the death rate due to the disease d_I set to 0.5.

For the seasonal variation, an example with A_0 set to 1.0 and $d_I = 0.5$ is shown in figure 3

Finally, figure 4 shows a simulation with the inclusion of vaccination with a vaccination rate of $f = 0.4$.

To benchmark the two solvers, the S, I, R-values were computed for varying number of grid points and Monte Carlo cycles, while the CPU time was recorded. The resulting log-log plot of the error/cpu time as a function of cpu time is shown in figure 5.

4 Analysis

All four populations have the same parameters apart from the rate of recovery, b, which is smallest for population A and largest for D. From this, we would expect that population D will have a larger resistance to the disease, as more people become immune faster. This is indeed what we see in the Runge Kutta simulation of the simple model with a constant population count. For populations A and B, we observe an initial spike in the number of infected people, followed by an increase in the number of recovered. For populations C and D, the number of infected people never increases, but goes down even from the start. This is exactly what we expected, as it is harder for the disease to spread when the rate of recovery is higher. Without the possibility of dying from the disease, the infected people eventually become recovered, before becoming susceptible again and stabilizing. In population C, we see that the disease has a slight relapse sometime between $t = 5.0$ and $t = 7.5$, and consequently the amount of susceptible people goes down, before stabilizing at around $t = 10$. In the Monte Carlo experiment, we generally observe the same pattern, though with the addition of a larger variance in the populations. This

[1] https://github.com/casparwb/FYS3150/tree/master/5_project

	Population			
	A	B	C	D
S	0.25	0.5	0.75	1.0
I	0.25	0.1	0.035	0.0
R	0.5	0.4	0.214	0.0

Tabell 1: The fraction equilibrium values for the simple model, for each population.

is especially apparent in population B and C, while populations A and D have a significantly smaller variance in the result. This is most likely caused by the fact that populations A and D have the extreme values of b, which means that they will reach their equilibrium faster, which is what we indeed observe, while populations B and C have oscillations caused by more instability due to the middle values of b.

With the addition of the vital dynamics, we see that now the constant population count $N = S + I + R$ is no longer constant. For populations A and B, we see that the disease is "winning", in that the total population count is decreasing. This means that the birth rate is not enough to keep the population constant or growing with the given recovery rate. This is not the case, however, for populations C and D, who both have a rise in their populations, with D having a steeper incline as a result of the higher rate of recovery. The Monte Carlo experiment has the same issues as the simple model, in that there are strong fluctuations in the population counts in B and C. Additionally, while the Monte Carlo implementation tends to oscillate around the "correct"values as computed with the Runge Kutta method, it does not necessarily manage to capture the smaller variations in the populations.

Seasonal variation was introduced by letting the rate of transmission oscillate as a cosine function. We set the period of the oscillation such that it would have two peaks within the given time frame. In the Runge Kutta solution in figure 3, we see the effect of this addition in mainly population A, B, and C, with only a slight visible difference in population D when compared to with only the vital dynamics. As we would expect, we see that the susceptible population oscillates with two peaks, causing oscillations in the other populations as well. Since we are varying the rate of transmission, we know that when this parameter increases, we should expect the number of infected people to also increase, which also causes more people to die. This is indeed what we see, especially in population B, where the total population count has a gradual negative slope throughout the simulation, but also has small intervals with a steeper slope, which correlates with a higher amount of infected people. In this interval we also see that the number of recovered people increases, which tells us that the total population count does not drop as much as it could due to the number of people who survive the disease and become immune. While the introduction of seasonal variation does produce some interesting visual result, it does not greatly affect the population count at the end of the simulation when compared to the result of the model with the same vital dynamic parameters in figure 2. This means that we would most likely have to increase the total time of the simulation to visualize possible changes between the final result of the two models. The results from the Monte Carlo simulation of the model with seasonal variation is very similar to the one with $\omega = 0$. This is not surprising, as as we mentioned earlier, the Monte Carlo method does not capture smaller fluctuations in the different groups, combined with the fact that the end result of the population counts are very similar to the simulation without seasonal variation. However, the variance in the N and S values of mainly population B and C are now even greater than in the previous model, which means that we can assume that having more fluctuations in the model, as caused by the introduction of seasonal variation, causes greater variance and error in the Monte Carlo simulation.

With the introduction of vaccination, as seen in figure 4, we see a drastic change change in the evolution of the different populations. In all populations apart from A, the total population count increases following a small dip. The vaccination causes the amount of infected people to never increase again once it reaches zero. For population A, however, the rate of recovery is so low that the effect of vaccination with a rate of 0.4 is not enough to keep the population from dying from both natural causes and the illness. Additionally, we see that the effect of seasonal variation of the rate of transmission is only visible in population A with the effect of vaccination included. This does not mean, however, that vaccination did not affect population A at all. At $T = 20$, the total population is about 100 more than with no vaccination. By doubling the vaccination rate to 0.8, it was observed that population A stabilized after around 5 time steps, at a total population count of 300, while the other populations maintained a visually identical evolution to that with $f = 0.4$. One of the most striking differences in the Monte Carlo implementation of the vaccination model, when compared to the previous model with seasonal variation, is the significant reduction in the fluctuations and variance of the different groups, mainly in populations B and C. This is not surprising, as the introduction of vaccination has produced significantly more stable results, particularly when compared to the seasonal variation model. As a result, the Monte Carlo results will be more reliable in this model than the results from the model with vital dynamics or seasonal variation.

In order to compare the two different methods of simulating the coupled differential equations, the error and CPU time produced when simulating the most simple model for a varying number of grid points and Monte Carlo cycles was plotted in figure 5. In this figure, we have visualized the error per CPU time as a function of the CPU time, which again is the function of number of grid points or Monte Carlo cycles. The results show that the Monte Carlo method is more unstable, showing significant jumps in the plot, while the Runge Kutta method performs much more smoothly for the varying number of grid points. We still observe that, in general, the two methods have a very similar incline, which means the error scales somewhat similarly when increasing the number of integration points or Monte Carlo cycles. However, the Monte Carlo method has a significantly worse starting point than the Runge Kutta implementation, which means that it requires a substantially larger amount of CPU time to produce the same results. In population A, we see that the Monte Carlo method produces an error of approximately $10^{1.5-2.5} = 10^{-1.5}$, while the Runge Kutta solver produced an error of $\approx 10^{-4.5-2.5} = 10^{-6.5}$, which is a substantial difference. With this result, it is hard to find a compelling argument for using the Monte Carlo method over the Runge Kutta iterative solver. Yes, by using Monte Carlo methods we can more accurately simulate the discrete transitions between the groups as is more realistic, but it is unsure how this affects the final result. Nevertheless, while the Monte Carlo method has some serious drawbacks, mainly that it requires a lot more CPU time to produce similarly accurate results as the Runge Kutta solver, it is still interesting to see how Monte Carlo methods can be used to solve coupled differential equations.

Suggestions for future experiments; run longer Monte Carlo simulations to see exactly how many cycles and CPU hours it would take to produce results as accurate as those produced by the Runge Kutta solver, and try more combinations of the different parameters, in addition to different values for the initial conditions. Additionally, a more realistic model can be produced by introducing a time-dependence on the vaccination rate.

5 Conclusion

In this experiment we have successfully produced two methods for simulating the spread of disease with using the SIRS model. The first method, using the Runge Kutta 4 algorithm, produced satisfactory results with small errors in the most simple model, and allowed us to accurately observe the evolution of the different groups as a function of time. In order to account for the realistic discrete transitions between the different groups, a Monte Carlo method was implemented. It was observed that fluctuations in the populations caused a greater variance in the results produced by the Monte Carlo solver, and thus it was deemed to not be accurate for certain parameter combinations. Additionally, the results did not show the smaller details and variations in the evolution of the populations. The Monte Carlo method also proved to be significantly slower in producing accurate results when compared to the Runge Kutta solver. Taking this into account, we can conclude that if having discrete transitions is crucial for an experiment, then the Monte Carlo method will produce good results with the additional cost of extra CPU time, while the Runge Kutta method is the better choice if this is not important, and more details want to be studied in how the populations change over time.

A Runge Kutta algorithm

For a single differential equation $y' = f(t, y)$ with the initial condition $y(t_0) = y_0$ where f is a given function. Given the number of steps N, maximum time T, and step size $\Delta t = T/N$, the Runge-Kutta method is given as

for $n = 0, 1, 2, ..., N - 1$ do

$$k_1 = f(t_n, y_n)\Delta t, \tag{11}$$

$$k_2 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_1}{2})\Delta t, \tag{12}$$

$$k_3 = f(t_n + \frac{\Delta t}{2}, y_n + \frac{k_2}{2})\Delta t, \tag{13}$$

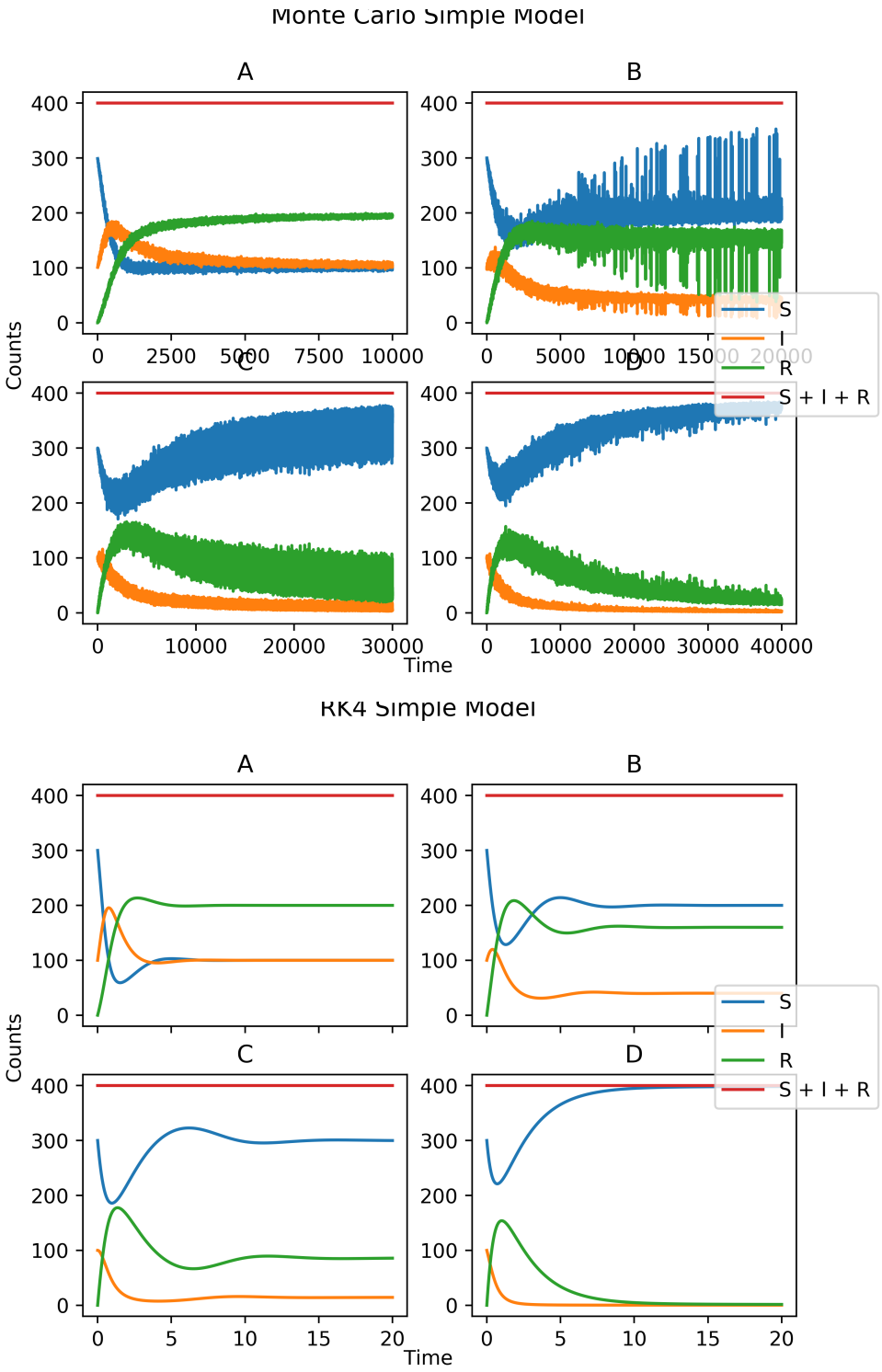
$$k_4 = f(t_n + \Delta t, y_n + k_3), \tag{14}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{15}$$

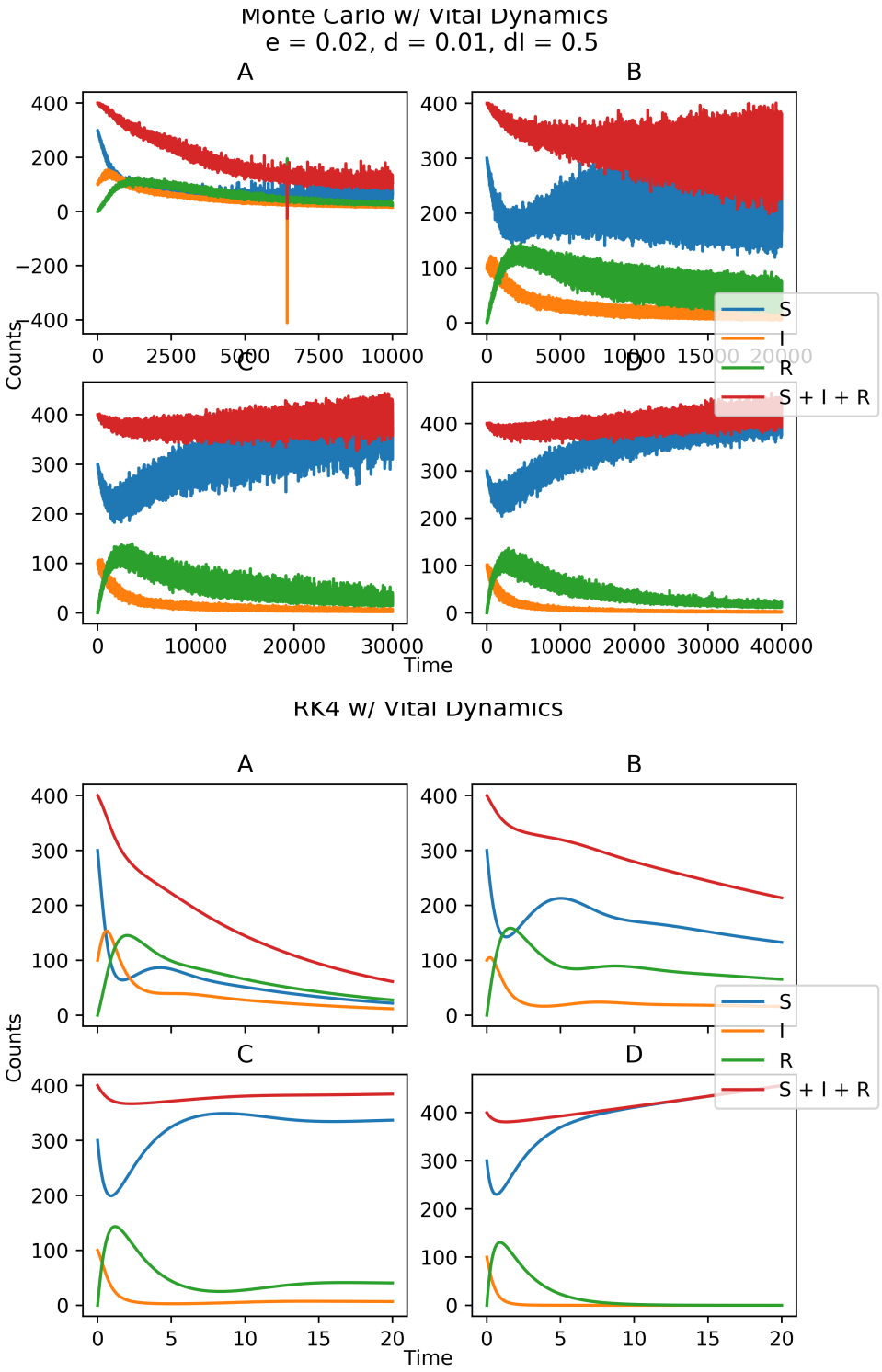
$$t_{n+1} = t_n + \Delta t. \tag{16}$$

end

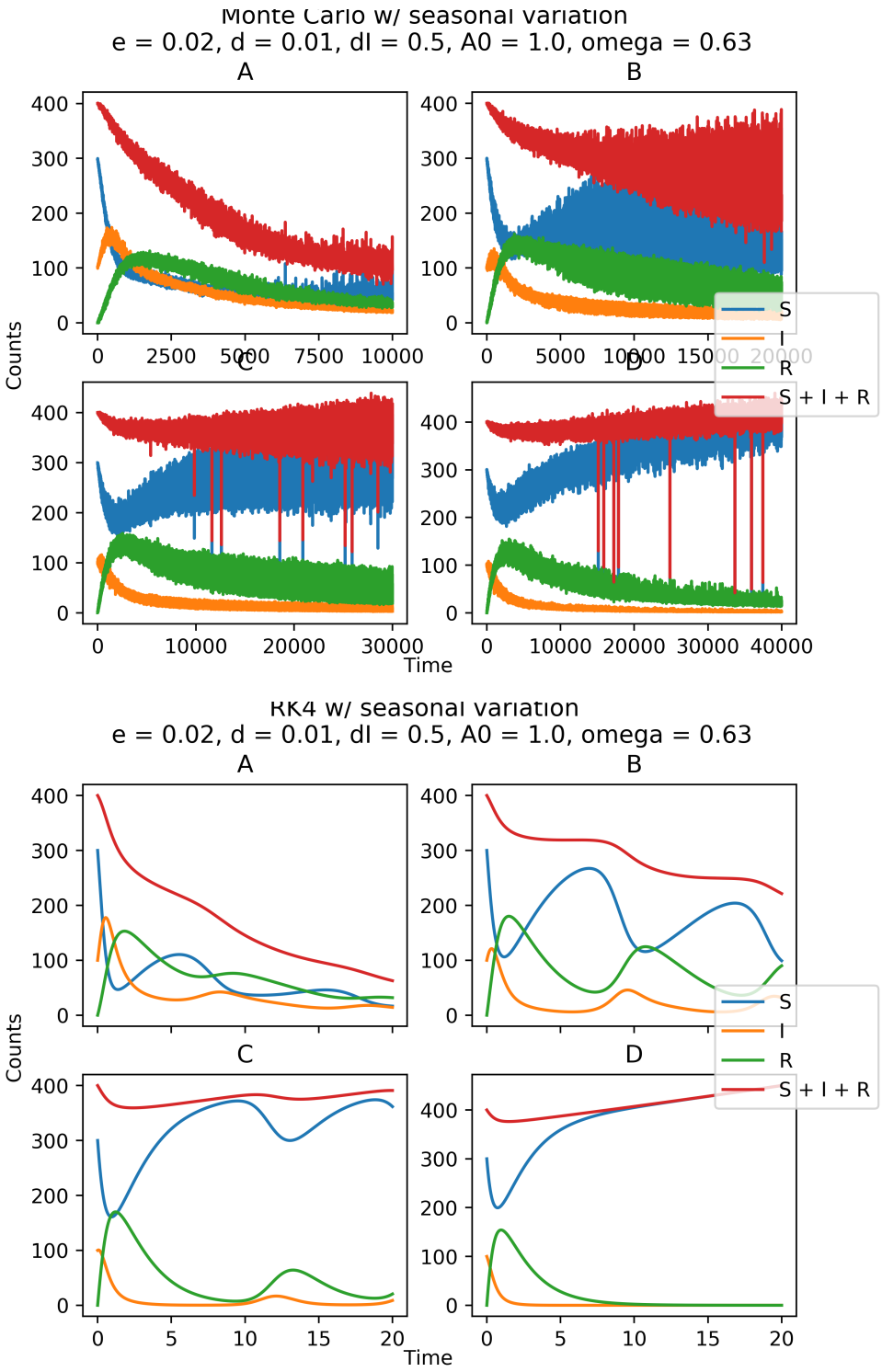
For coupled differential equations, a vector u is sent to the function f, where u contains the differential equations to solve.



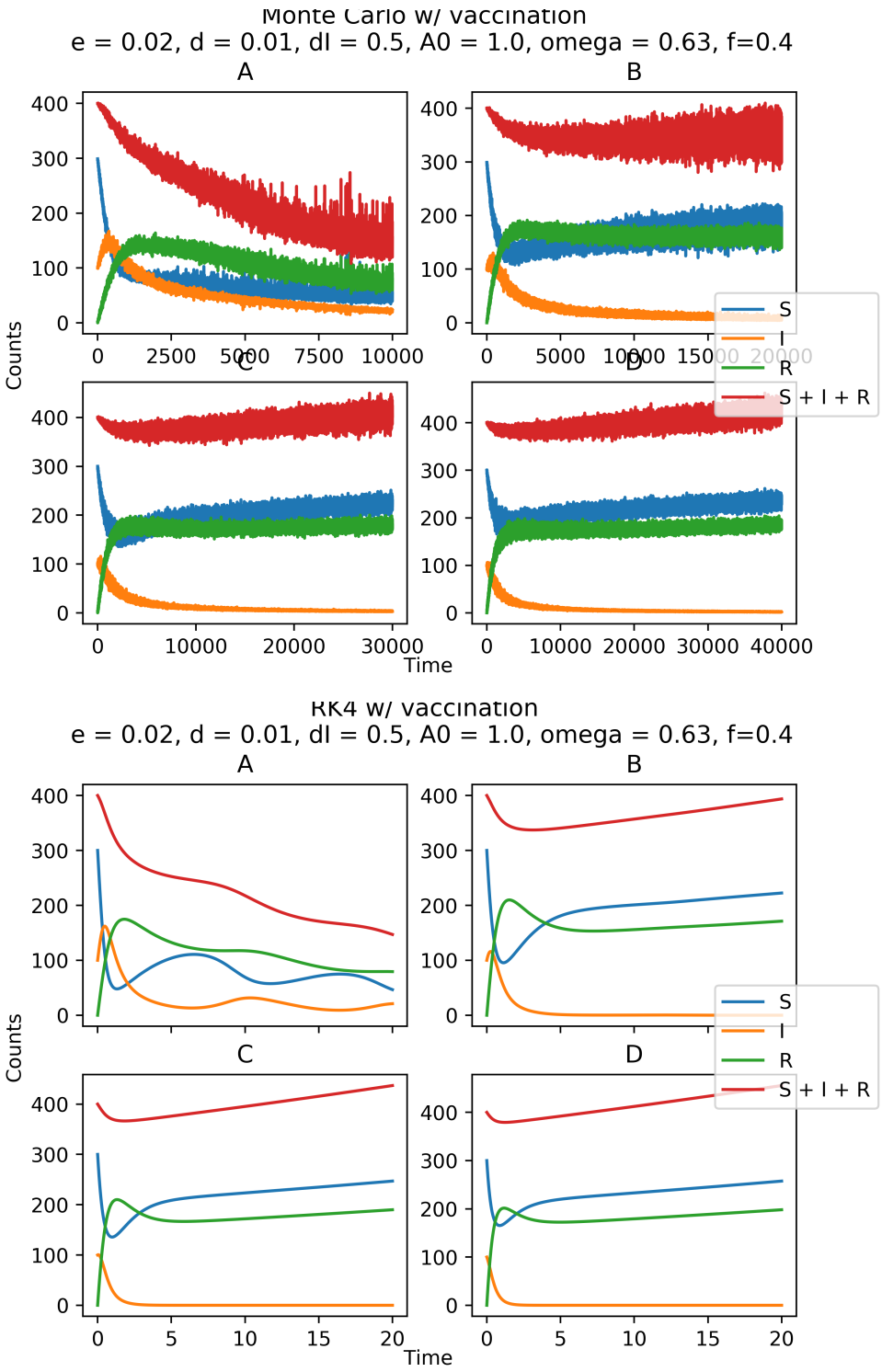
Figur 1: Caption for this figure with two images



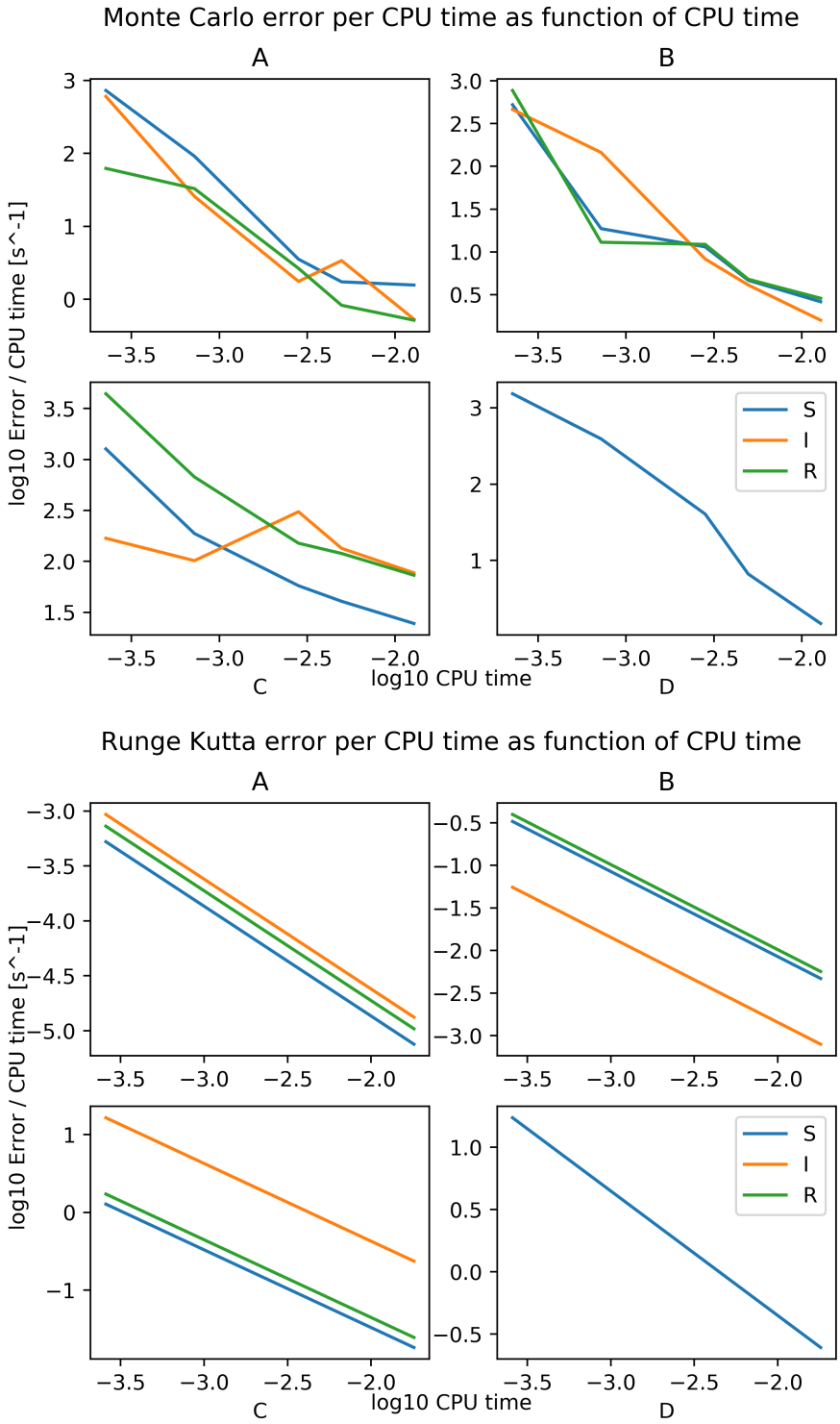
Figur 2: Caption for this figure with two images



Figur 3: Caption for this figure with two images



Figur 4: Caption for this figure with two images



Figur 5: Caption for this figure with two images