Integrating ordinary differential equations in R

Aaron A. King
with contributions from
Ben Bolker, John Drake, Pej Rohani, and Dave Smith

October 30, 2017

1 Introduction

Here we begin our study of computational techniques for studying epidemiological models. In this session we introduce the numerical solution (or integration) of nonlinear differential equations using the sophisticated solvers found in the package deSolve. Numerical integration is one of the most important tools we have for the analysis of epidemiological models.

2 The SIR model

As we saw in the lecture, the classical SIR compartmental model divides a population of hosts into three classes: susceptible, infected, recovered. The model describes how the fraction of a population in each of these classes changes with time. Alternatively, the model can track the number of individuals in each class. Births are modeled as flows from "nowhere" into the susceptible class; deaths are modeled as flows from the S, I, or R compartment into "nowhere". If S, I, and R refer to the fractions of indivduals in each compartment, then these *state variables* change according to the following system of differential equations:

$$\begin{aligned}
\frac{dS}{dt} &= B - \lambda\,S - \mu\,S\\
\frac{dI}{dt} &= \lambda\,S - \gamma\,I - \mu\,I\\
\frac{dR}{dt} &= \gamma\,I - \mu\,R
\end{aligned}$$

Here, B is the crude birth rate (births per unit time), μ is the death rate and γ is the recovery rate. We'll assume that the force of infection, λ, has the form

$$\lambda = \beta\, I$$

so that the risk of infection a susceptible faces is proportional to the *prevalence* (the fraction of the population that is infected). This is known as the assumption of frequency-dependent transmission.

3 Solving ODEs in R

Like almost all epidemiological models, one can't solve these equations analytically. However, we can compute the *trajectories* of a continuous-time model such as this one by integrating the equations numerically. Doing this accurately involves a lot of calculation, and there are smart ways and not-so-smart ways of going about it. This very common problem has been very thoroughly studied by numerical analysts for generations so that, when the equations are smooth, well-behaved functions, excellent numerical integration algorithms are readily available to compute approximate solutions to high precision. In particular, R has several sophisticated ODE solvers which (for many problems) will give highly accurate solutions. These algorithms are flexible, automatically perform checks, and give informative errors and warnings. To use the numerical differential equation solver package, we load the deSolve package

```r
require(deSolve)
```

[Here, you may get an error saying that the deSolve package is not installed. If you do, run the following command:

```r
install.packages("deSolve")
```

] The ODE solver we'll use is called `ode`. Let's have a look at the help page for this function.

```r
?ode
```

`ode` needs to know the *initial values* of the state variables (`y`), the `times` at which we want solutions, the right-hand side of the ODE `func`. The latter can optionally depend on some parameters (`parms`).

4 SIR for a closed epidemic

Let's study the SIR model for a closed population, i.e., one in which we can neglect births and deaths. Recall that the differential equations for the closed epidemic are

$$\begin{aligned}\frac{dS}{dt} &= -\beta\,S\,I\\ \frac{dI}{dt} &= \beta\,S\,I-\gamma\,I\\ \frac{dR}{dt} &= \gamma\,I\end{aligned}$$

To encode these equations in a form suitable for use as the `func` argument to `ode`, we'll need to write a function. For example:

```r
closed.sir.model <- function (t, x, params) {
  ## first extract the state variables
  S <- x[1]
  I <- x[2]
  R <- x[3]
  ## now extract the parameters
  beta <- params["beta"]
  gamma <- params["gamma"]
  ## now code the model equations
  dSdt <- -beta*S*I
  dIdt <- beta*S*I-gamma*I
  dRdt <- gamma*I
  ## combine results into a single vector
  dxdt <- c(dSdt,dIdt,dRdt)
  ## return result as a list!
  list(dxdt)
}
```

Note that the order and type of the arguments and output of this function must exactly match `ode`'s expectations. Thus, for instance, the time variable `t` must be the first argument even if, as is the case here, nothing in the function depends on time. [When the RHS of the ODE are independent of time, we say the ODE are *autonomous*.] Note also, that `ode` expects the values of the ODE RHS to be the first element of a `list`.

Now we can call `ode` to compute trajectories of the model. To do this, we'll need some values of the parameters. If we're thinking of a disease something like measles, and measuring time in years, we might use something like:

```r
params <- c(beta=400,gamma=365/13)
```

Exercise 1. What is the infectious period of this disease? What is R_0 in this case?

We now state the times at which we want solutions and specify the *initial conditions*, i.e., the starting values of the state variables S, I, and R:

```
times <- seq(from=0,to=60/365,by=1/365/4) # returns a sequence
xstart <- c(S=0.999,I=0.001,R=0.000)      # initial conditions
```

Next, we compute a model trajectory with the `ode` command and store the result in a data-frame:

```
out <- as.data.frame(
                     ode(
                         func=closed.sir.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )
```

and plot the results in Figure 1 using the commands:

```
plot(I~time,data=out,type='l')
```

Exercise 2. Suppose that you'd rather measure time in days. Modify the parameters accordingly and verify your modifications.

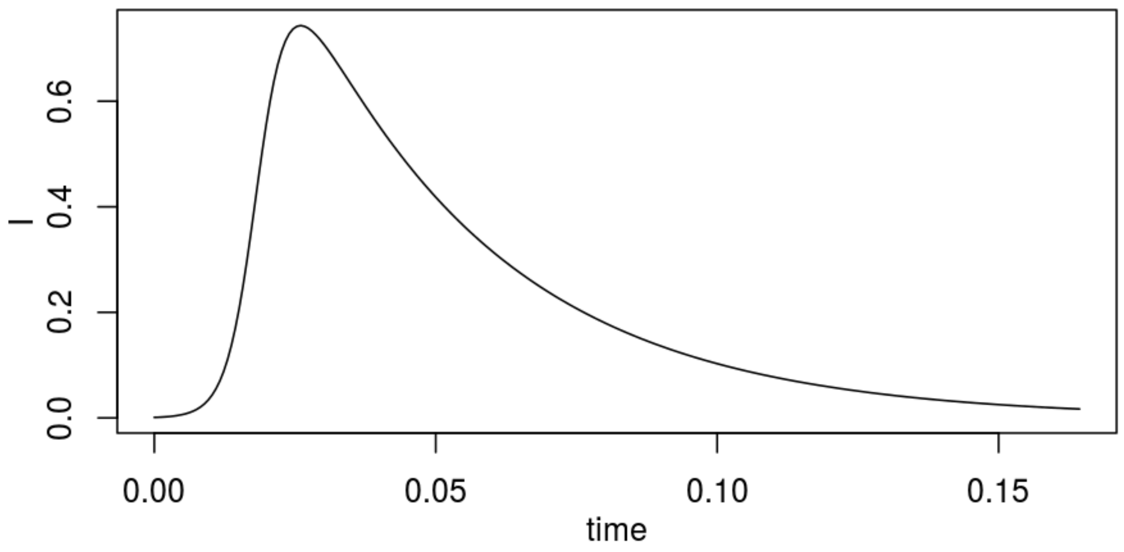


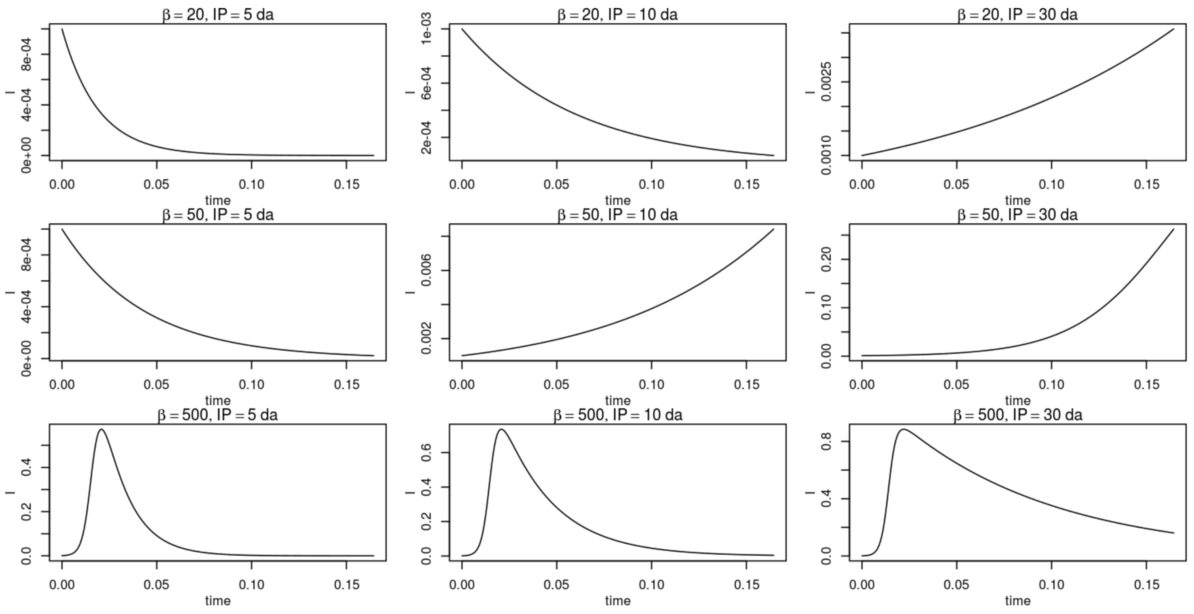
Figure 1: Trajectory of the SIR model of an epidemic in a closed population. I is the fraction of the population infected.

Let's study how the epidemic curve depends on the transmission rate, β, and the infectious period. In particular, we'll investigate how the epidemic curve changes as we vary β from 20 to 500 and the infectious period from 5 to 30 days.

```r
betavals <- c(20,50,500)
ips <- c(5,10,30)
gammavals <- 365/ips

## set some plot parameters
op <- par(mgp=c(2,1,0),mar=c(3,3,1,1),mfrow=c(3,3))

for (beta in betavals) {
  for (gamma in gammavals) {
    params <- c(beta=beta,gamma=gamma)
    out <- as.data.frame(
                           ode(
                               func=closed.sir.model,
                               y=xstart,
                               times=times,
                               parms=params
                               )
                           )
    title <- bquote(list(beta==.(beta),"IP"==.(365/gamma)~"da"))
    plot(I~time,data=out,type='l',main=title)
  }
}
```



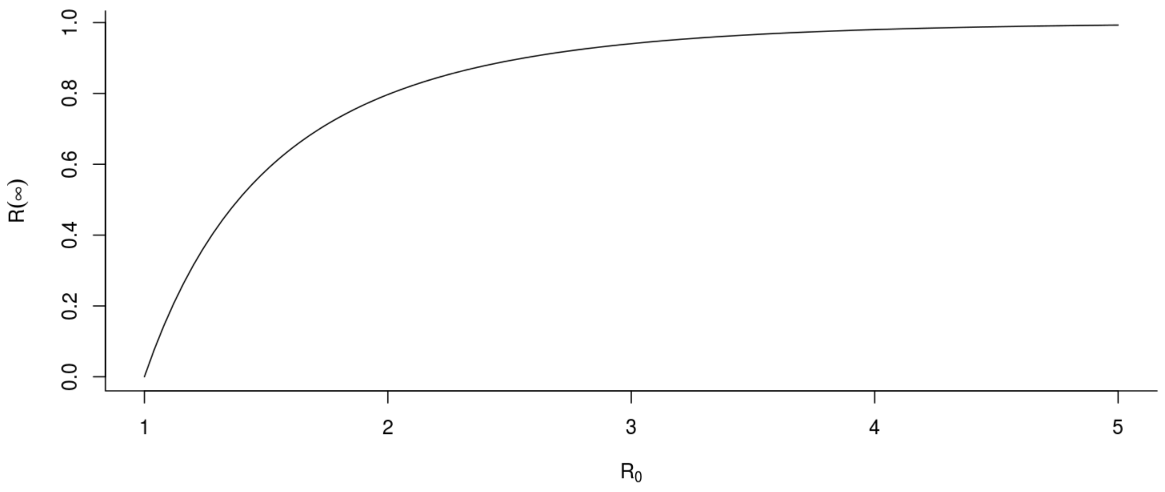


Figure 2: The final size, $R(\infty)$, of an SIR epidemic depends only on R_0.

```
par(op)       # restore old settings
```

Simulation is a useful tool, but its power is limited. The next exercise demonstrates the importance of being able to analyze the equations as well.

Exercise 3. For each of the above parameter combinations, describe the system's behavior. Compute R_0 for each parameter combination and relate it to the behavior of the system.

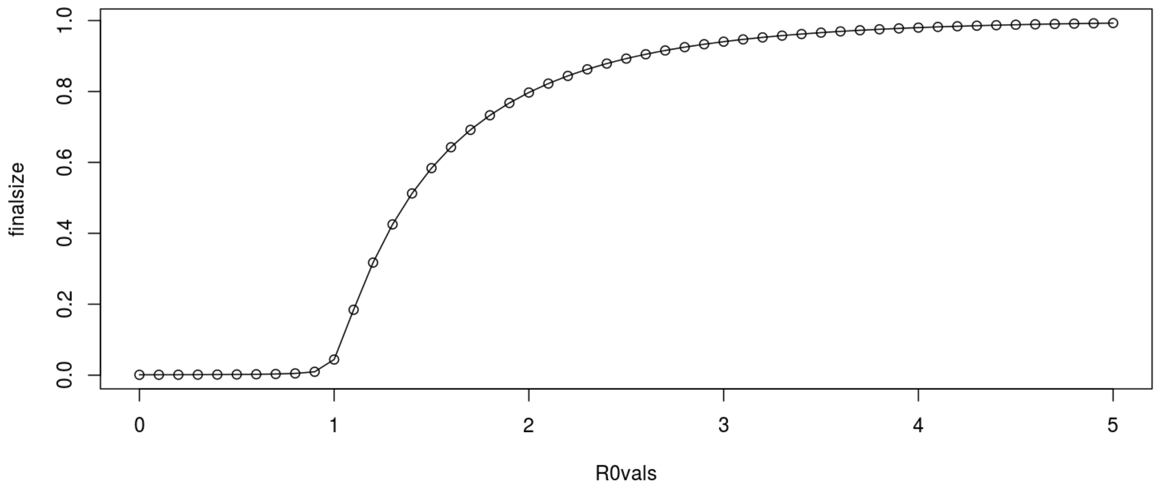
Exercise 4. Use the ODE solver to study the dependence of the epidemic's *final size* on R_0. Compare your results with the predictions of the final size equation

$$1 - R(\infty) = S(0)\, e^{-R(\infty)\, R_0} = e^{-R(\infty)\, R_0}$$

solutions of which are plotted in Fig. 2.

Solution. Here's one code that will solve the exercise. Others are certainly possible.

```
times <- seq(from=0,to=20,by=1/12) # you might think 20 yr would be enough!
xstart <- c(S=0.999,I=0.001,R=0.000)
R0vals <- seq(0,5,by=0.1)
gamma <- 365/13
betavals <- R0vals*gamma                # R0 = beta/gamma
finalsize <- numeric(length(R0vals))    # a vector to hold the solutions
for (k in seq_along(betavals)) {
    params <- c(beta=betavals[k],gamma=gamma)
    out <- as.data.frame(
                         ode(
                             func=closed.sir.model,
                             y=xstart,
                             times=times,
                             parms=params
                             )
                         )
    finalsize[k] <- tail(out$R,1)        # the final value of R
}
plot(finalsize~R0vals,type='o')
```



5 SIR dynamics in an open population

Over a sufficiently short time scale, the assumption that the population is closed is reasonable. To capture the dynamics over the longer term, we'll need to account for births and deaths, i.e., allow the population to be an *open* one. As we've seen, if we further assume that the birth rate equals the death rate, then the SIR equations become

$$\begin{aligned}\frac{dS}{dt} &= \mu - \beta\,S\,I - \mu\,S\\ \frac{dI}{dt} &= \beta\,S\,I - \gamma\,I - \mu\,I\\ \frac{dR}{dt} &= \gamma\,I - \mu\,R\end{aligned}$$

We must modify the ODE function accordingly:

```r
open.sir.model <- function (t, x, params) {
  beta <- params["beta"]
  mu <- params["mu"]
  gamma <- params["gamma"]
  dSdt <- mu*(1-x[1])-beta*x[1]*x[2]
  dIdt <- beta*x[1]*x[2]-(mu+gamma)*x[2]
  dRdt <- gamma*x[2]-mu*x[3]
  list(c(dSdt,dIdt,dRdt))
}
```

We'll need to specify a birth/death rate in addition to the two parameters we specified before:

```
params <- c(mu=1/50,beta=400,gamma=365/13)
```

We integrate the equations as before:

```
times <- seq(from=0,to=25,by=1/365)
out <- as.data.frame(
                     ode(
                         func=open.sir.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )
```

We can plot each of the state variables against time, and I against S, as shown in Fig. 3, using the commands:

```
op <- par(fig=c(0,1,0,1),mfrow=c(2,2),
          mar=c(3,3,1,1),mgp=c(2,1,0))
plot(S~time,data=out,type='l',log='y')
plot(I~time,data=out,type='l',log='y')
plot(R~time,data=out,type='l',log='y')
plot(I~S,data=out,log='xy',pch='.',cex=0.5)
par(op)
```

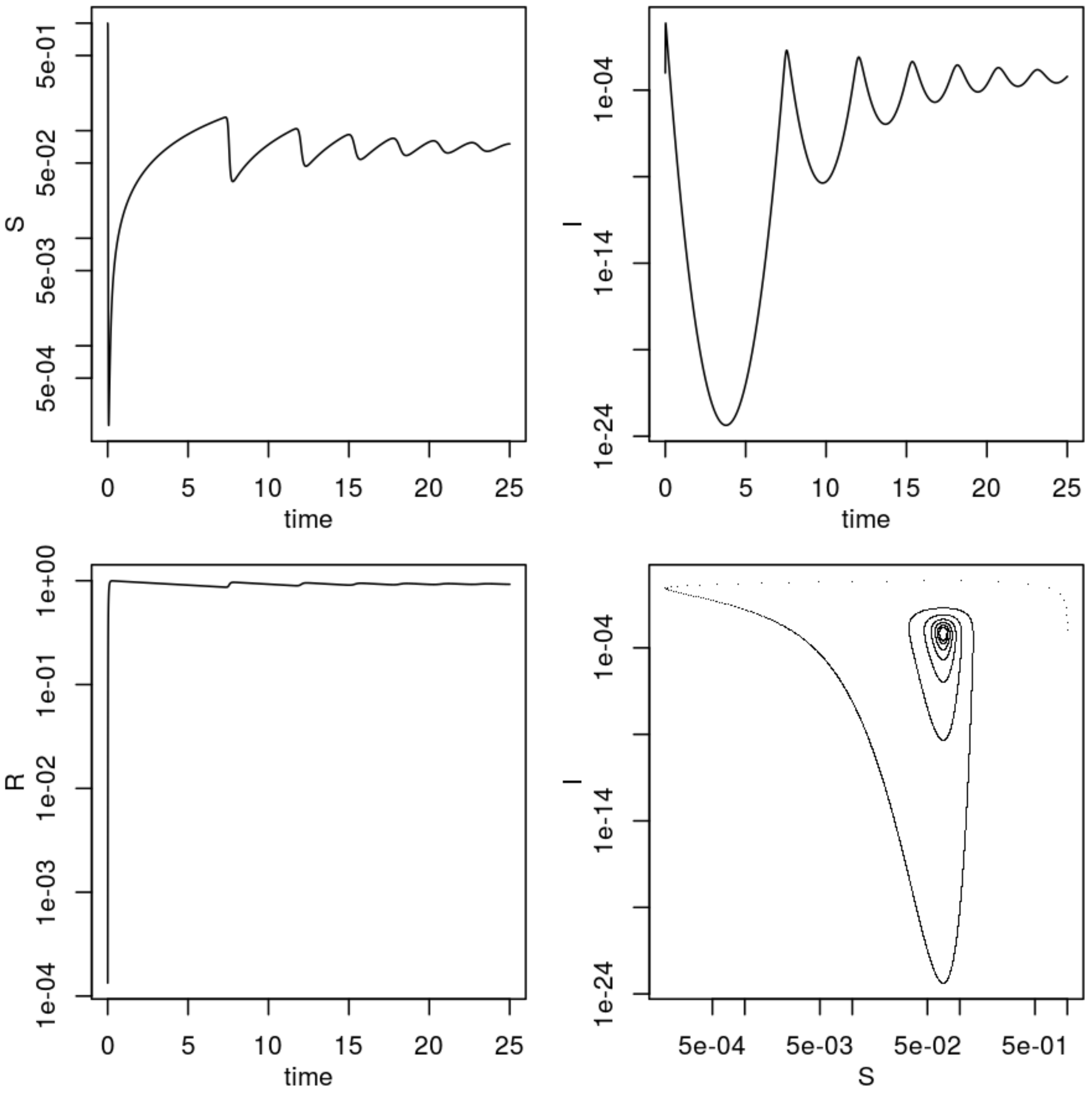


Figure 3: Trajectory of the SIR model. Each state variable is plotted against time, and I is also plotted against S.

Exercise 5. Explore the dynamics of the system for different values of the β and γ parameters by simulating and plotting trajectories as time series and in phase space (e.g., I vs. S). Use the same values of β and γ we looked at above. How does the value of R_0 affect the results?

Exercise 6. Under the assumptions of this model, the average host lifespan is $1/\mu$. Explore how host lifespan affects the dynamics by integrating the differential equations for lifespans of 20 and 200 years.

The compartmental modeling strategy can be put to use in modeling a tremendous range of infections. The following exercises make some first steps in this direction.

***Exercise 7.** The SIR model assumes lifelong sterilizing immunity following infection. For many infections, immunity is not permanent. Make a compartment diagram for an SIRS model, in which individuals lose their immunity after some time. Write the corresponding differential equations and modify the above codes to study its dynamics. Compare the SIR and SIRS dynamics for the parameters $\mu = 1/50$, $\gamma = 365/13$, $\beta = 400$ and assuming that, in the SIRS model, immunity lasts for 10 years.

***Exercise 8.** Make a diagram, write the equations, and study the dynamics of the SEIR model for the dynamics of an infection with a latent period. Compare the dynamics of SIR and SEIR models for the parameters $\mu = 1/50$, $\gamma = 365/5$, $\beta = 1000$ and assuming that, in the SEIR model, the latent period has duration 8 days.

6 Nonautonomous equations

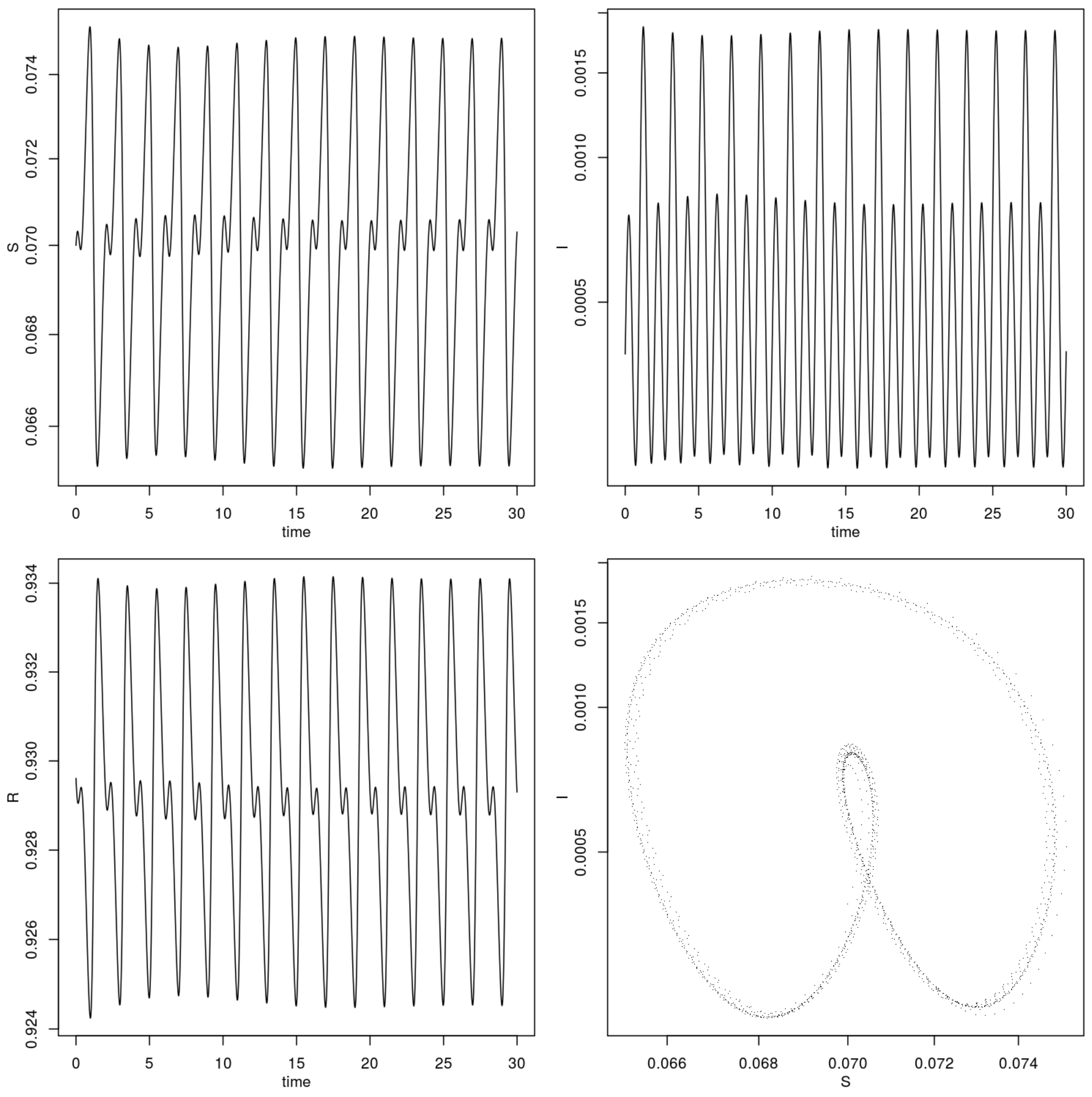
SIR with seasonal transmission

The simple SIR model always predicts damped oscillations towards an equilibrium (or pathogen extinction if R_0 is too small). This is at odds with the recurrent outbreaks seen in many real pathogens. Sustained oscillations require some additional drivers in the model. An important driver in childhood infections of humans (e.g., measles) is seasonality in contact rates because of aggregation of children the during school term. We can analyze the consequences of this by assuming sinusoidal forcing on β according to $\beta(t) = \beta_0 \,(1 + \beta_1 \cos(2\,\pi\,t))$. We can modify the code presented above to solve the equations for a seasonally forced epidemic.

```r
seas.sir.model <- function (t, x, params) {
  beta0 <- params["beta0"]
  beta1 <- params["beta1"]
  mu <- params["mu"]
  gamma <- params["gamma"]
  beta <- beta0*(1+beta1*cos(2*pi*t))
  dSdt <- mu*(1-x[1])-beta*x[1]*x[2]
  dIdt <- beta*x[1]*x[2]-(mu+gamma)*x[2]
  dRdt <- gamma*x[2]-mu*x[3]
  list(c(dSdt,dIdt,dRdt))
}

params <- c(mu=1/50,beta0=400,beta1=0.15,gamma=365/13)
xstart <- c(S=0.07,I=0.00039,R=0.92961)
times <- seq(from=0,to=30,by=7/365)
out <- as.data.frame(
                     ode(
                         func=seas.sir.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )

op <- par(fig=c(0,1,0,1),mfrow=c(2,2),
          mar=c(3,3,1,1),mgp=c(2,1,0))
plot(S~time,data=out,type='l',log='y')
plot(I~time,data=out,type='l',log='y')
plot(R~time,data=out,type='l',log='y')
plot(I~S,data=out,log='xy',pch='.',cex=0.5)
```



```r
par(op)
```

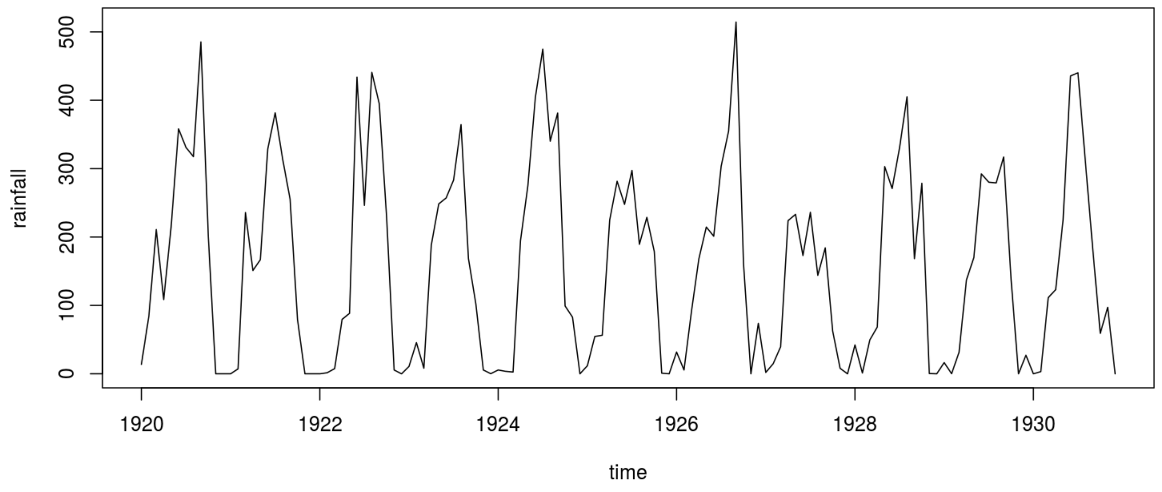
Exercise 9. Explore the dynamics of the seasonally forced SIR model for increasing amplitude β_1. Be sure to distinguish between transient and asymptotic dynamics.

Forcing with a covariate

When a covariate forces the equations, we must interpolate the covariate. To give an example, let's suppose that the transmission rate depends on rainfall, $R(t)$, and that we have data on rainfall (in mm/mo).

```r
rain <- read.csv("https://kingaa.github.io/thid/data/dacca_rainfall.csv")
rain$time <- with(rain,year+(month-1)/12)

plot(rainfall~time,data=rain,type='l')
```



Let's assume that transmission depends on rainfall according to

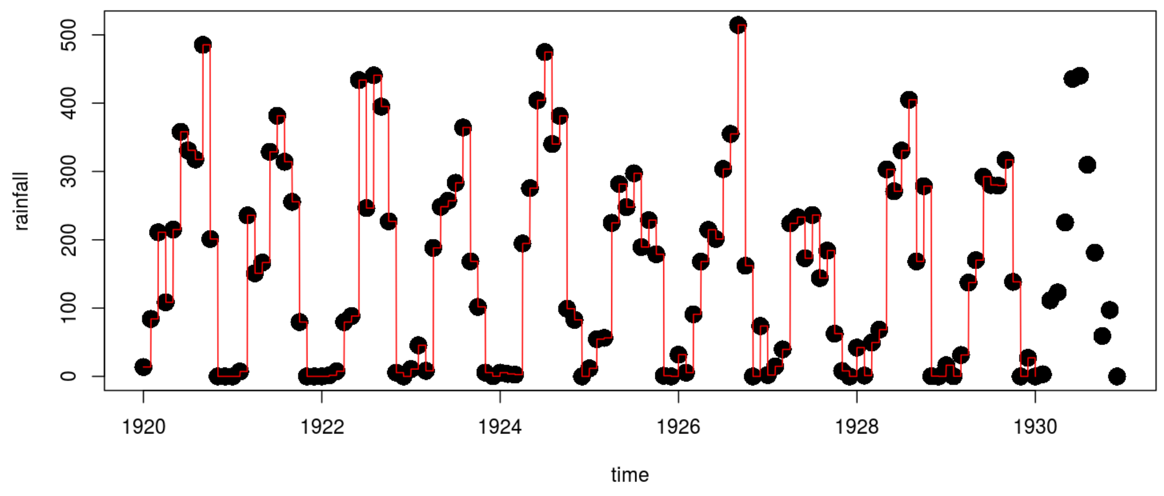
$$\log \beta(t) = \frac{a\,R(t)}{b + R(t)}$$

Since the data are accumulated monthly rainfall figures but the ODE integrator will need to evaluate $R(t)$ at arbitrary times, we'll need some way of interpolating the rainfall data. R affords us numerous ways of doing this. The R functions `approxfun` and `splinefun` construct interpolating functions in different ways; see the documentation on these functions.

```r
interpol <- with(rain,approxfun(time,rainfall,rule=2,method='constant'))

data.frame(time=seq(from=1920,to=1930,by=1/365)) -> smoothed.rain
smoothed.rain$rainfall <- sapply(smoothed.rain$time,interpol)

plot(rainfall~time,data=rain,col='black',cex=2,pch=16,log='')
lines(rainfall~time,data=smoothed.rain,col='red')
```

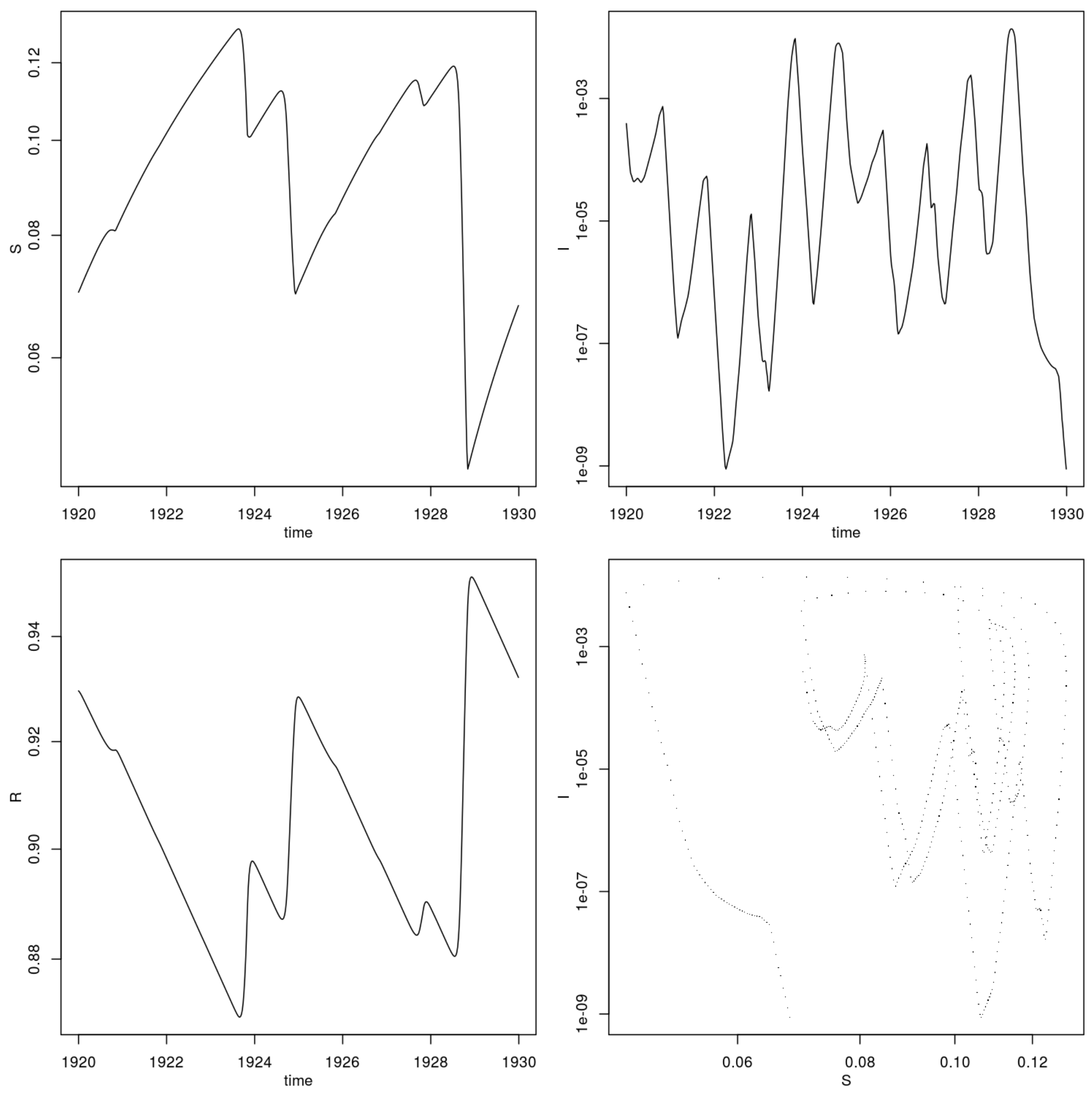


```
rain.sir.model <- function (t, x, params) {
  a <- params["a"]
  b <- params["b"]
  mu <- params["mu"]
  gamma <- params["gamma"]
  R <- interpol(t)
  beta <- a*R/(b+R)
  dSdt <- mu*(1-x[1])-beta*x[1]*x[2]
  dIdt <- beta*x[1]*x[2]-(mu+gamma)*x[2]
  dRdt <- gamma*x[2]-mu*x[3]
  list(c(dSdt,dIdt,dRdt))
}

params <- c(a=500,b=50,mu=1/50,gamma=365/13)
xstart <- c(S=0.07,I=0.00039,R=0.92961)
times <- seq(from=1920,to=1930,by=7/365)
out <- as.data.frame(
                     ode(
                         func=rain.sir.model,
                         y=xstart,
                         times=times,
                         parms=params
                         )
                     )

op <- par(fig=c(0,1,0,1),mfrow=c(2,2),
          mar=c(3,3,1,1),mgp=c(2,1,0))
plot(S~time,data=out,type='l',log='y')
```

```r
plot(I~time,data=out,type='l',log='y')
plot(R~time,data=out,type='l',log='y')
plot(I~S,data=out,log='xy',pch='.',cex=1)
```



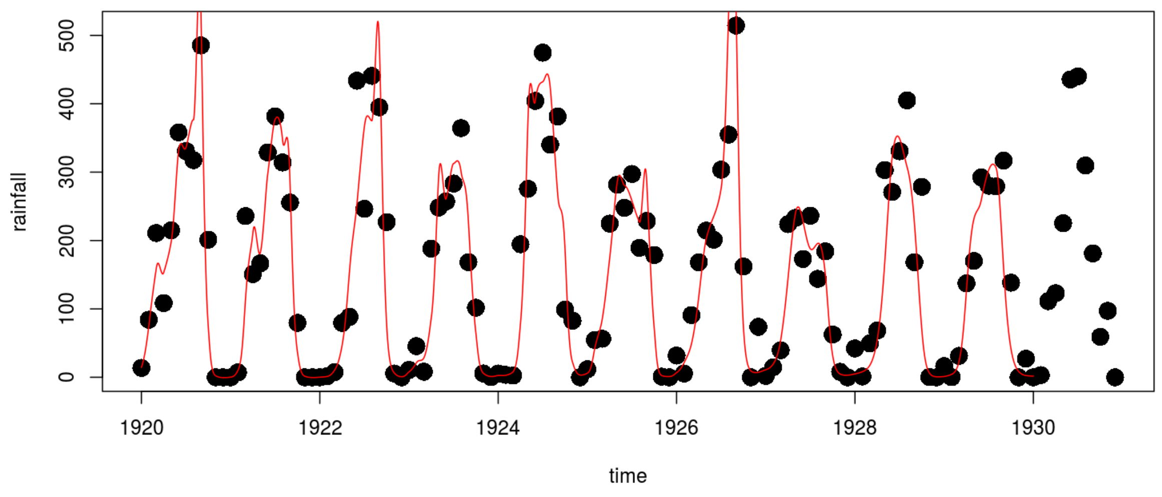
```r
par(op)
```

More generally, any fitting method that has an associated `predict` method can be used. For example, we can use local polynomial regression, `loess`, to smooth the rainfall data. Do `?loess` and `?predict.loess` for more details.

```r
fit <- loess(log1p(rainfall)~time,data=rain,span=0.05)

data.frame(time=seq(from=1920,to=1930,by=1/365)) -> smoothed.rain
smoothed.rain$rainfall <- expm1(predict(fit,newdata=smoothed.rain))

plot(rainfall~time,data=rain,col='black',cex=2,pch=16,log='')
lines(rainfall~time,data=smoothed.rain,col='red')
```


rainfall
time