

```

## -- Attaching packages ----- tidyverse
1.3.0 --
## v ggplot2 3.3.0    v purrr 0.3.3
## v tibble 3.0.0     v dplyr 0.8.5
## v tidyr 1.0.0      v stringr 1.4.0
## v readr 1.3.1      v forcats 0.4.0
## -- Conflicts ----- tidyverse_conflicts()
--
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
##
## Attaching package: 'magrittr'
## The following object is masked from 'package:purrr':
##
##   set_names
## The following object is masked from 'package:tidyr':
##
##   extract
##
## Attaching package: 'lubridate'
## The following object is masked from 'package:base':
##
##   date
## Loading required package: usethis
##
## Attaching package: 'foreach'
## The following objects are masked from 'package:purrr':
##
##   accumulate, when
##
## Attaching package: 'incidence'
## The following object is masked from 'package:broom':
##
##   bootstrap

```

Epidemiology – Human, Disease Vectors, Water Borne Illnesses

Marc Los Huertos

April 2, 2020

1 Introduction

1.1 Disease, Pandemics, and Mathematical Epidemiology

COVID-19 began rapidly spreading around the world in late 2019. And by the spring, Italy went into lock down, California declared a state of emergency, schools and universities around the globe suspended in person classes and events, and businesses reduced travel and pushed tele-work policies. All of this was designed to slow the spread of the disease. These efforts are broadly referred to as social distancing.

1.2 Disease Spread and Control

The contagious nature of diseases has been appreciated for hundreds, if not thousands of years. However, until 'germ' theory as articulated by Louis Pasteur, the mechanism that imparted infections was highly speculative.

1.3 Value of Social Distancing

The idea is to reduce person-to-person contact to make spreading the disease less likely. The effects of this are often illustrated in images such as those in the chart below, where the red plot is flattened to spread out the disease as much as possible. This helps to ensure that there are sufficient resources available for a sick population, which will help improve survival rates.

Flattening the curve to keep infection manageable (Source: Fast.ai). How do we determine the value of such distancing strategies and model this spread?
<https://rviews.rstudio.com/2020/03/19/simulating-covid-19-interventions-with-r/>

1.4 Modeling Disease – SIR

Mathematical models can simulate the effects of a disease at many levels, ranging from how the disease influences the interactions between cells in a single patient

(within-host models) to how it spreads across several geographically separated populations (metapopulation models).

Models simulating disease spread within and among populations, such as those used to forecast the COVID-19 outbreak, are typically based on the Susceptible - Infectious - Recovered (SIR) framework.

The SIR model is a compartmental model for modeling how a disease spreads through a population. It's an acronym for Susceptible, Infected, Recovered. When a disease is introduced to a population, the people move from one of these classes (or compartments) to the next. When they reach the R state, they're no longer able to be infected, depending on your interpretation, they either survived the disease and are now immune or succumbed to the illness and are out of the population.

SIR models are compartmental disease models. "Susceptible", "Infectious", and "Recovered" are compartments, and each individual in the population (N) is assigned to one of these compartments. To unpack this a bit further:

Susceptible individuals have no immunity to the disease (immunity can come from prior exposure, vaccination, or a mutation that confers resistance). Therefore, they can become infected. Susceptible individuals can move into the "Infectious" compartment through contact with an infectious person.

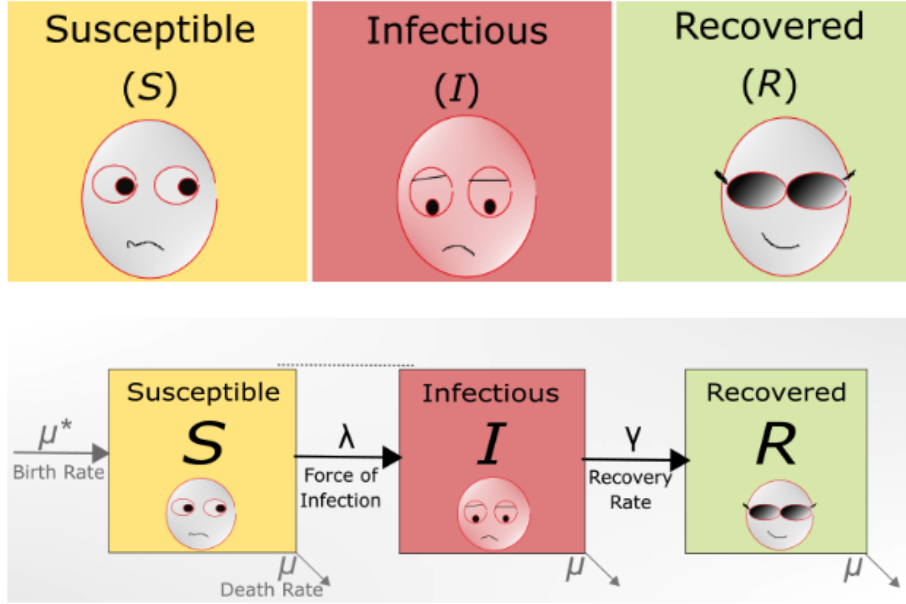
Infectious people have the disease and can spread it to others. Infectious individuals can move into the "Recovered" compartment by recovering from the illness.

Recovered individuals can no longer become infected, typically because they have immunity from a prior exposure. Many SIR-based models assume that a recovered person remains immune, which is often appropriate if immunity is long-lasting (e.g., chicken pox) or the disease is being modeled over a relatively short time period.

Because people move between compartments, the number of people in each compartment changes over time. The SIR model captures population changes in each compartment with a system of ordinary differential equations (ODEs) to model the progression of a disease.

The standard SIR model can be schematically represented as: SIR framework.png

The Kermack-McKendrick model is an SIR model for the number of people infected with a contagious illness in a closed population over time. It was proposed to explain the rapid rise and fall in the number of infected patients observed in epidemics such as the plague (London 1665-1666, Bombay 1906) and cholera (London 1865). It assumes that the population size is fixed (i.e., no births, deaths due to disease, or deaths by natural causes), incubation period of the infectious agent is instantaneous, and duration of infectivity is same as length of the disease. It also assumes a completely homogeneous population with no age, spatial, or social structure.



The model consists of a system of three coupled nonlinear ordinary differential equations,

$$\frac{dS}{dt} = -\beta SI/N \quad (1)$$

$$\frac{dI}{dt} = \beta SI/N - \gamma I \quad (2)$$

$$\frac{dR}{dt} = \gamma I, \quad (3)$$

where t is time, $S(t)$ is the number of susceptible people, $I(t)$ is the number of people infected, $R(t)$ is the number of people who have recovered and developed immunity to the infection.

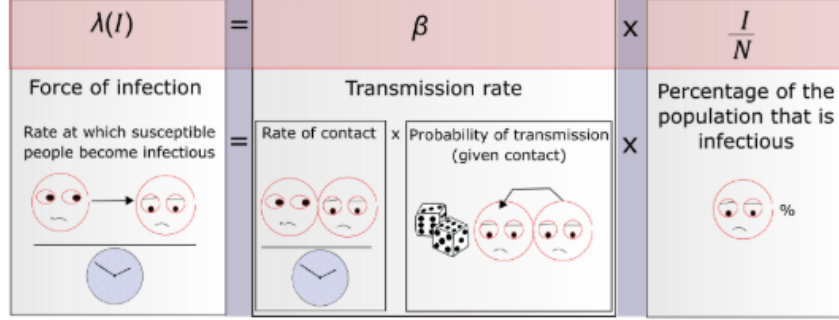
We have three ordinary differential equations in the time domain with three parameters: α , β , γ .

- α is the inverse of the incubation period ($1/t_{incubation}$)
- β is the average contact rate in the population or infection rate.
- γ is the inverse of the mean infectious period ($1/t_{infectious}$) or recovery rate.

This system is non-linear, however it is possible to derive its analytic solution in closed-form. Other numerical tools include Monte Carlo methods.

First, note that from:

$$\lambda(I) = \beta \frac{I}{N} .$$



$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dR}{dt} = 0, \quad (4)$$

it follows that:

$$S(t) + I(t) + R(t) = \text{constant} = N, \quad (5)$$

The key value governing the time evolution of these equations is the so-called epidemiological threshold,

$$R_0 = (\beta S)/\gamma \quad (6)$$

Note that the choice of the notation R_0 is a bit unfortunate, since it has nothing to do with R used for recovery. R_0 is defined as the number of secondary infections caused by a single primary infection; in other words, it determines the number of people infected by contact with a single infected person before their death or recovery.

1.5 Modeling COVID-19 w/SIR with R







We walk through a SIR epidemiological model and simulate it with R.

First, we need to estimate the parameters:











- Rate of Transmission, $\beta = 0.5944$
- Rate of Recovery, $\gamma = 1/\text{average time of infection (22 days)} = 0.045$

β is derived by assuming that the basic reproduction number, $R_0 = \frac{\beta}{\theta} \cdot \frac{\sigma}{\sigma + \mu} = 2.8$ (referring to Imai et al., 2020, Riou and Althaus, 2020, J.T. Wu et al., 2020, Zhao et al., 2020a, Zhao et al., 2020b) when $\alpha=0$, by using the next generation matrix approach (van den Driessche and Watmough, 2002). The time unit is in year if not mentioned.







$$\frac{dS}{dt} = -\lambda(I)S$$

$\frac{dS}{dt}$	= -	$\lambda(I)$	X	S
Rate of change of the number of susceptible people	= -	Force of infection The rate at which susceptible people become infectious	X	Number of susceptible people
#  — 		 →  — 		# 

$$\frac{dI}{dt} = \lambda(I)S - \gamma I$$

$\frac{dI}{dt}$	=	$\lambda(I)$	X	S	-	γ	X	I
Rate of change of the number of infectious people	=	Force of infection Rate at which susceptible people become infectious	X	Number of susceptible people	-	Recovery Rate Rate at which infectious people recover	X	Number of infectious people
#  — 		 →  — 		# 		 →  — 		# 

$$\frac{dR}{dt} = \gamma I$$

$\frac{dR}{dt}$	=	γ	X	I
Rate of change of the number of recovered people	=	Recovery rate Rate at which infectious people recover	X	Number of infectious people
#  — 		 →  — 		# 

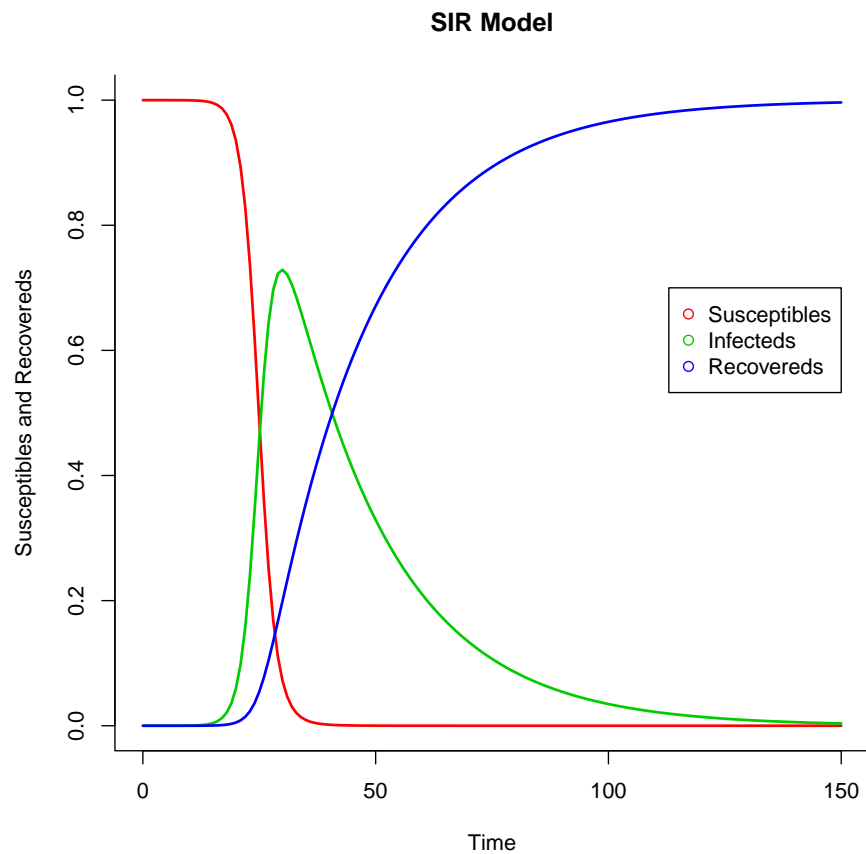
```

sir <- function(time, state, parameters) {
  with(as.list(c(state, parameters)), {
    dS <- -beta * S * I
    dI <- beta * S * I - gamma * I
    dR <- gamma * I

    return(list(c(dS, dI, dR)))
  })
}

init <- c(S = 1-1e-6, I = 1e-6, 0.0)
parameters <- c(beta, gamma)
times <- seq(0, 150, by = 1)
out <- as.data.frame(ode(y = init, times = times, func = sir, parms = parameters))
out$time <- NULL

```



Parameter	Model 1
β_0	0.1
k	10
T_e	7
T_i	10.25

1.6 Modeling Disease – SEIR

The first model is the basic SEIR without social distancing, then we add social distancing to show how the potential effectiveness of these strategies.

An extension of the classic SIR model and simply adds one more equation to show those who are exposed,

$$\frac{dS}{dt} = -\beta SI \quad (7)$$

$$\frac{dE}{dt} = \beta SI - \omega E \quad (8)$$

$$\frac{dI}{dt} = \omega E - \gamma I \quad (9)$$

Equation 1 is the change in people susceptible to the disease and is moderated by the number of infected people and their contact with the infected. Equation ?? gives the people who have been exposed to the disease. It grows based on the contact rate and decreases based on the incubation period whereby people then become infected.

1.7 SEIR with R

In the case of β is defined as $\beta_0 k$, where β_0 is the probability of infection or exposure and k is the frequency of exposure. ω is the coefficient of migration of latency and is estimated as $1/T_e$, where T_e is the average latency. Finally, γ , which is $1/T_i$ is the recovery rate.

```
SEIR <- function(beta0, k, Te, Ti){
  # Function to return derivatives of SEIR model
  seir_ode<-function(t,Y,par){
    S<-Y[1]
    E<-Y[2]
    I<-Y[3]
    R<-Y[4]

    beta<-par[1]
    omega<-par[2]
    gamma<-par[3]
```



```

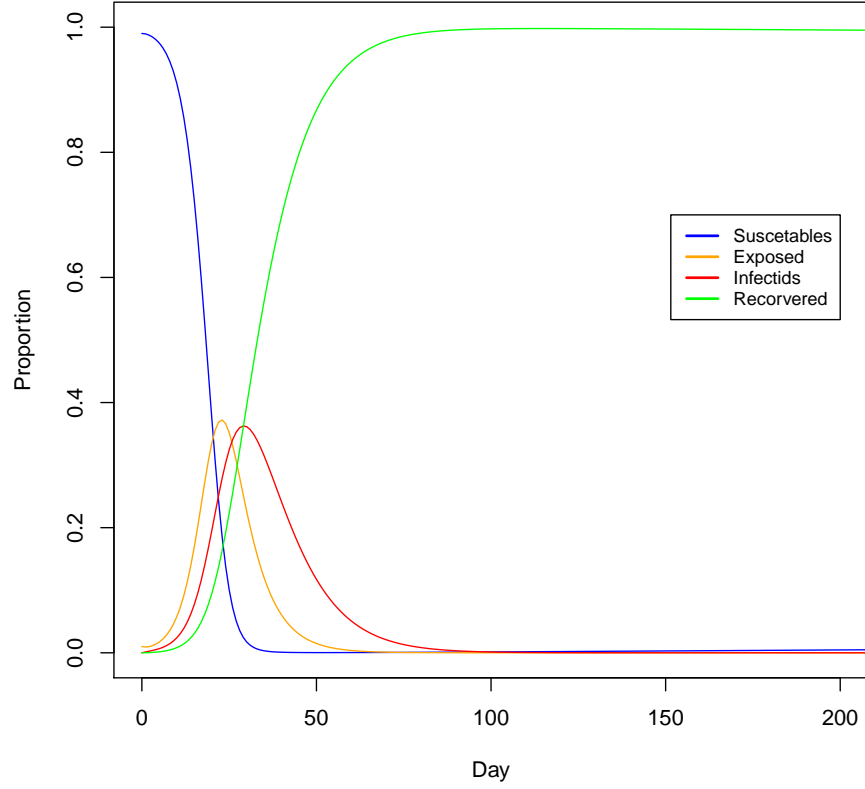
mu<-par[4]

dYdt<-vector(length=3)
dYdt[1]=mu-beta*I*S-mu*S
dYdt[2]=beta*I*S-(omega+mu)*E
dYdt[3]=omega*E-(gamma+mu)*I

return(list(dYdt))
}
# Set parameter values
beta = beta0*k
#beta<-520/365;
omega = 1/Te
# sigma<-1/60;
gamma = 1/Ti
# gamma<-1/30;
mu<-774835/(65640000*365) # UK birth and population figures 2016
# mu=1000000
init<-c(0.99,0.01,0.0)
t<-seq(0,365)
par<-c(beta,omega,gamma,mu)
# Solve system using lsoda
lsoda(init,t,seir_ode,par)
}

Model1 <- SEIR(.1, 10, 7, 10.25)

```



1.8 SEIR-SEI and Disease Vectors

When diseases are spread by vectors with their own dynamics, for example malaria and mosquitos, we should add additional compartments, namely susceptibility, exposure, and infection compartments for the mosquito. Mosquitos are not born with the malaria parasite, but must be infected from a host, so we can now include additional ODEs:

$$\frac{dS_v}{dt} = -\beta_v S_v I_v / N \quad (10)$$

$$\frac{dI_v}{dt} = \beta S_v I_v / N - \gamma I_v \quad (11)$$

$$\frac{dR_v}{dt} = \gamma I_v, \quad (12)$$

The key value governing the time evolution of these equations is the so-called epidemiological threshold and I think this can be used to link these two models, but I don't know how to yet.

$$R_0 = ? \quad (13)$$

```
# Function to return derivatives of SEIR model
seir_ode<-function(t,Y,par){
# Human Population
  S<-Y[1]
  E<-Y[2]
  I<-Y[3]
  R<-Y[4]

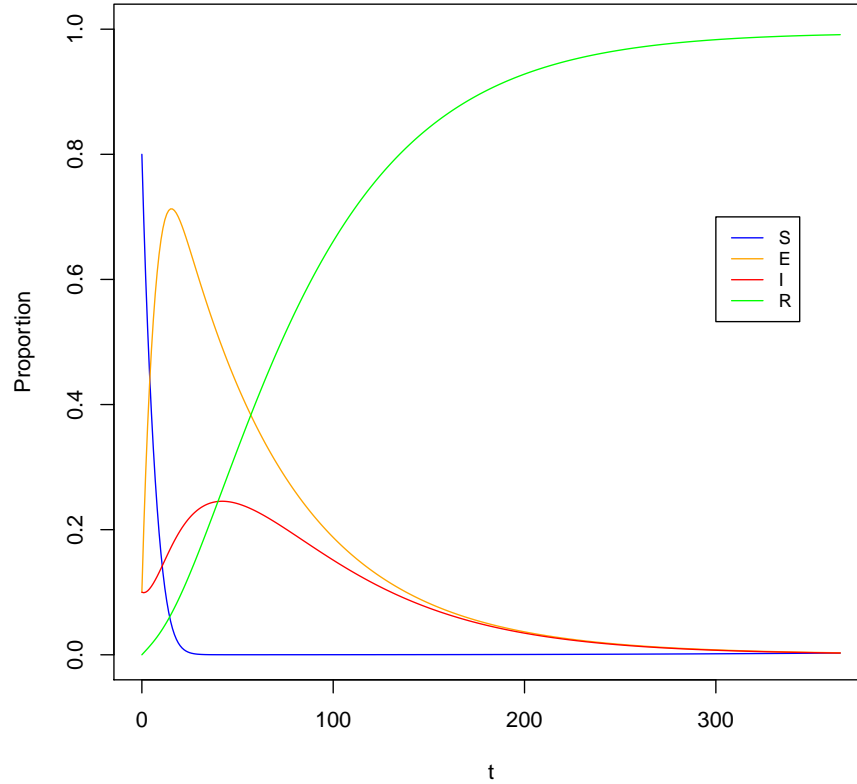
  beta<-par[1]
  sigma<-par[2]
  gamma<-par[3]
  mu<-par[4]

  dYdt<-vector(length=3)
  dYdt[1]=mu-beta*I*S-mu*S
  dYdt[2]=beta*I*S-(sigma+mu)*E
  dYdt[3]=sigma*E-(gamma+mu)*I

  return(list(dYdt))
}

# Set parameter values
beta<-520/365;
sigma<-1/60;
gamma<-1/30;
mu<-774835/(65640000*365) # UK birth and population figures 2016
init<-c(0.8,0.1,0.1)
t<-seq(0,365)
par<-c(beta,sigma,gamma,mu)
# Solve system using lsoda
sol<-lsoda(init,t,seir_ode,par)

# Plot solution
plot(t,sol[,2],type="l",col="blue",ylim=c(0,1),ylab="Proportion")
lines(t,sol[,3],col="orange")
lines(t,sol[,4],col="red")
lines(t,1-rowSums(sol[,2:4]),col="green")
legend(300,0.7,legend=c("S","E","I","R"),col=c("blue","orange","red","green"), lty=1, cex=0.8)
```



2 Model with Spatial Structure

2.1 Model Structure

The model relies on spatially explicit movements of 'subjects' that basically bounce around in a box, i.e. petri dish. Each subject is assigned a random location and trajectory (angle) of movement and velocity of movement vector. I based the model on an Washington Post article by Harry Stevens (March 14, 2020). Although his model, written in Java and is probably much cleaner, I wanted to create something that students could use to adjust parameters and evaluate different policy strategies, using a simple visualization as a foundation.

2.2 Building a Model

To track subject movements, contact with others, etc I have defined an array to track subject, thus, have to do some backflips to track overlap at each time

step to see if people are sharing space and pathogens.

I'm not a programmer, so this is going to need some TLC as it's pretty inefficient.

2.3 Model Parameterization

I have define the following parameters for the model, which will be further described in each section below.

	Model	N	Velocity	tstep	Stationary	Inf_Dist	init_infact	asympt	symp	rec
1	Model1	50.00	5.00	2000.00	0.00	5.00	2.00	24.00	96.00	720.00
2	Model2	100.00	5.00	2000.00	0.00	5.00	2.00	24.00	96.00	720.00
3	Model3	100.00	5.00	3000.00	0.40	5.00	2.00	24.00	96.00	720.00
4	Model4	100.00	5.00	3000.00	0.40	5.00	2.00	60.00	60.00	720.00
5	TBD									720.00
6	Test	5.00	5.00	20.00	0.00	300.00	1.00	24.00	96.00	720.00

But for now, I'd like to see how these align with the equations described above.

2.4 Model Functions

First, functions are useful when you have repetitive actions to make – in this case, I have defined functions to move subjects based on their speed and direction of travel.

Since these functions rely on the data array within another function (thus, not globally available, until it is 'returned' at the end of the function), I had to include a reference to the data in the function so that the functions could find the array.

```
#create function to move the subject
move_x = function(data_array, s, t){
  data_array[s, 4, t] * cos(data_array[s,3, t]*pi/180)}
move_y = function(data_array, s, t){
  data_array[s, 4, t] * sin(data_array[s,3, t]*pi/180)}

# check function
# move_y(data_arr, 1, 1); move_x(data_arr, 1,1)

# move function
move = function(data_array, s, t){
  c(data_array[s, 1, t-1]+ move_x(data_array, s, t-1),
    data_array[s, 2, t-1]+ move_y(data_array, s, t-1),
    data_array[s, 3, t-1],
    data_array[s, 4, t-1],
    data_array[s, 5, t-1],
```

```

    data_array[s, 6, t-1])
  }

  # Test functions
  # data_arr[1,,1]
  # move(data_arr, 1, 2)
  # Model1; i = 2

# Contact Infection Function
contact = function(data_array, t, Infect_dist){

  # testfunction
  #data_array=Model1_50; t=88; infect_dist = 5
  pairs = cbind(as.vector(data_array[, "x", t]),
                 as.vector(data_array[, "y", t])); pairs
  tmp = as.matrix(dist(pairs, method = "euclidean" )); tmp
  tmp[upper.tri(tmp, diag = TRUE)] <- NA; tmp
  pairs <- which(tmp < Infect_dist ,arr.ind = TRUE);pairs
  tmp2 = cbind(as.numeric(rownames(tmp)[pairs[, 1]]),
               as.numeric(colnames(tmp)[pairs[, 2]])); tmp2
  contacts = dim(tmp2)[1]; contacts
  if(contacts>0){
    for(i in 1:contacts){
      # If one is exposed, susceptible is exposed
      if(data_array[tmp2[i,1],5,t]==2 & data_array[tmp2[i,2],5,t]==1){
        data_array[tmp2[i,2],5,t] = 2
      }
      # If one is exposed, susceptible is exposed
      if(data_array[tmp2[i,2],5,t]==2 & data_array[tmp2[i,1],5,t]==1){
        data_array[tmp2[i,1],5,t] = 2
      }
      # If one is symptomatic, the susceptible is exposed
      if(data_array[tmp2[i,1],5,t]==3 & data_array[tmp2[i,2],5,t]==1){
        data_array[tmp2[i,2], 5, t] = 2
      }
      # If one is symptomatic, the susceptible is exposed
      if(data_array[tmp2[i,1],5,t]==1 & data_array[tmp2[i,2],5,t]==3){
        data_array[tmp2[i,1], 5, t] = 2
      }
    }
  }

  # data_array[, , t]
  return(data_array)
}

```

```
# print(data_array[,5,t])
} # close function

#contact(Model1, 87, 5)
```

Next, I'm going to figure out how to create a function to test for euclidian distances to spread disease, but this will take some more effort ...

2.5 Model 1: Modeling Susceptability and Infections with No Behavior Changes and Recovery

For this model, people get instantenously sick and start spreading it around – not very realistic because when people get sick they are usually quarantined and stop bouncing around the environment. Nevertheless this sets up a good base line, where the poputation gets ill pretty quickly.

```
MoveSubjects = function(m=0){
  # m = 0
  N = params[m, 2]
  Velocity = params[m, 3]
  tstep = params[m, 4]
  Stationary = params[m,5];
  Infect_dist=params[m, 6];
  Init_infect=params[m, 7]
  asymp = params[m,8]
  symp = params[m,9]
  rec = params[m,10]

  # Define Data Array
  location = 6 # number of parameters to track
  data_arr = array(dim=c(N, location, tstep))
  dimnames(data_arr)[[2]] <- c("x", "y", "theta", "velocity", "status", "count")
  data_arr[, ,1]
  # Initialize Start Locations
  set.seed(2763)
  subj_x = runif(N, coord_range[1], coord_range[2]); subj_x
  subj_y = runif(N, coord_range[3], coord_range[4]); subj_y

  # Set Up Population Characteristics and Behaviors
  sheltered = sample(N*Stationary); sheltered
  velocity = rep(Velocity,N); velocity[sheltered]=0; velocity
  theta = round(runif(N, 0, 360),0); theta
  SEIR = rep(1, N); SEIR[sheltered] = 0
  SEIR[sample(N, Init_infect)] = 3; SEIR
  count = rep(0, N)
```

```

# Initial Locations
data_arr[:,1] = c(subj_x, subj_y, theta, velocity, SEIR, count)
data_arr[:,1]
# t = 2; s=1
for(t in 2:tstep){
  for(s in 1:N){
    # move subjects based on theta and speed
    data_arr[s,,t] = move(data_arr, s, t)
    # coarse corrections when hitting a boundary
    # Min x-boundary
    if(data_arr[s,1,t] < coord_range[1]){
      data_arr[s,3,t-1]=180-data_arr[s,3,t-1]
      data_arr[s,,t] = move(data_arr, s, t)
    }
    #Max x-boundary
    if(data_arr[s,1,t] > coord_range[2]){
      data_arr[s,3,t-1]=180-data_arr[s,3,t-1]
      data_arr[s,,t] = move(data_arr, s, t)
    }
    #Min y-boundary
    if(data_arr[s,2,t] < coord_range[3]){
      data_arr[s,3,t-1]=360-data_arr[s,3,t-1]
      data_arr[s,,t] = move(data_arr, s, t)
    }
    #Max y-boundary
    if(data_arr[s,2,t] > coord_range[4]){
      data_arr[s,3,t-1]=360-data_arr[s,3,t-1]
      data_arr[s,,t] = move(data_arr, s, t)
    }
    # Change Status based on Time Steps
    data_arr[s,6,t] = data_arr[s,6,t] + 1
    if(data_arr[s,5,t]==1 | data_arr[s,5,t]==0) data_arr[s,6,t]=0 # Reset Susceptables
    # Exposed becomes symptomatic
    if(data_arr[s,5,t]==2 & data_arr[s,6,t] > asymp){
      data_arr[s,5,t]=3}
    # Symptomatics move very little
    if(data_arr[s,5,t]==3 & data_arr[s,6,t] > symp){
      data_arr[s,4,t] = Velocity/2}
    # Time to Recovery
    if(data_arr[s,5,t]==3 & data_arr[s,6,t] > rec){
      data_arr[s,5,t] = 4
      data_arr[s,4,t] = Velocity
    }
  } # End of Subj Moving
}

```



```

# Time Step Proceesing
# Test for Euclidean Distances Here!

data_arr <- contact(data_arr, t, Infect_dist)

} # End of Time Step
return(data_arr)
}

```

2.6 Model Runs

For after running the models, I visually evaluated the models to make sure there weren't unintended consequences, i.e. usually from coding errors. Below are the models that I ran and the estimated run times.

```

library(tictoc)
#N=3, tstep=4, speed=5, init_infect=1, stationary = 0)
params

tic("Model0")
Test<- MoveSubjects(m=6)
toc() # .02s

tic("Model1")
Model1<- MoveSubjects(m=1)
toc() # 900s; 314s

tic("Model2")
Model2<- MoveSubjects(m=2)
toc() #1800s; 1158s; 1313s

tic("Model3")
Model3 <- MoveSubjects(m=3)
toc() #3308; 2437s; 2599s

tic("Model4")
Model4 <- MoveSubjects(m=4)
toc() #3308s; 2332s; 2616s

```

3 Visualizations

3.1 Tracking the Disease Transfer

```

#load("/home/CAMPUS/mwl04747/github/beginnersluck.RData")
# Susceptible 1, Exposed-Asymptomatic 2, Symptomatic 3, Recovered 4
SEIRcol = c("steelblue", "orange", "red", "green")

trackresults=function(model,m,tstep){
  results = data.frame(Days = NA, Sh = NA, S=NA, E=NA, I=NA, R=NA)
  for(t in 1:tstep){
    days = t/24
    sh = sum(as.vector(model[,5,t])==0)
    s = sum(as.vector(model[,5,t])==1, sh)
    e = sum(as.vector(model[,5,t])==2)
    i = sum(as.vector(model[,5,t])==3)
    r = sum(as.vector(model[,5,t])==4)
    results[t,] <- data.frame(Days = days, Sh = sh, S=s, E=e, I=i, R=r)
  }
  head(results)
  par(mfrow=c(1,1), mar=c(4, 5, 3, 2)+.1, las=1)
  plot(S~Days, data=results, ty='l', col=SEIRcol[1], ylab="", ylim=c(0,params[m,2]), xlab="Day")
  lines(E~Days, data=results, col=SEIRcol[2], lwd=2)
  lines(I~Days, data=results, col=SEIRcol[3], lwd=2)
  lines(R~Days, data=results, col=SEIRcol[4], lwd=2)
}

m = 6; trackresults(Test, m, params[m,4])

png(filename =
  "/home/CAMPUS/mwl04747/github/beginnersluck/Epidemiology/Model1.png");
m = 1; trackresults(Model1, m, 2000); dev.off()
png(filename =
  "/home/CAMPUS/mwl04747/github/beginnersluck/Epidemiology/Model2.png");
m = 2; trackresults(Model2, m, 2000); dev.off()
png(filename =
  "/home/CAMPUS/mwl04747/github/beginnersluck/Epidemiology/Model3.png");
m = 3; trackresults(Model3, m, 2000); dev.off()
png(filename =
  "/home/CAMPUS/mwl04747/github/beginnersluck/Epidemiology/Model4.png");
m = 4; trackresults(Model4, m, 2000); dev.off()

```

3.2 Static Spatial Plots

Because we are looking at a dynamic system, I don't find the static plot all that useful.

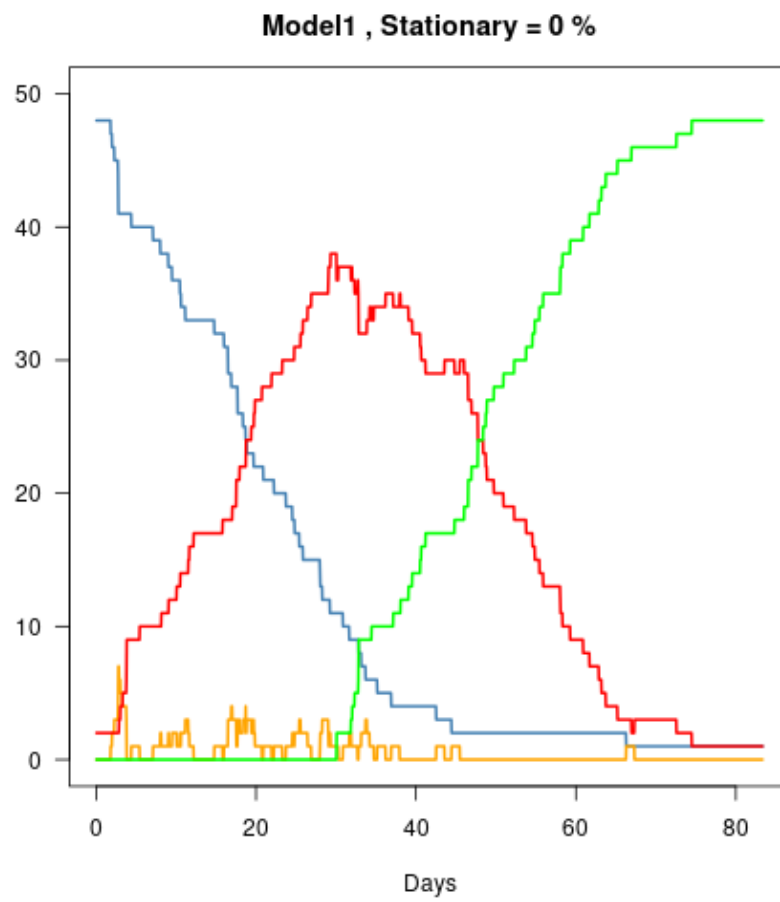


Figure 1: $N = 50$

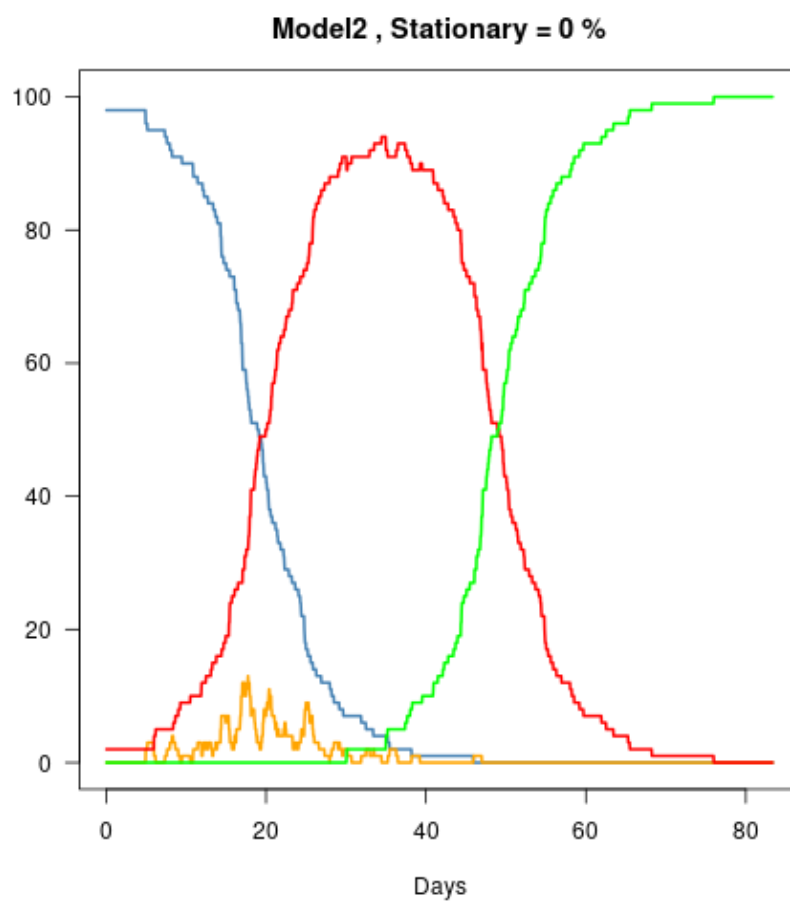


Figure 2: $N=100$

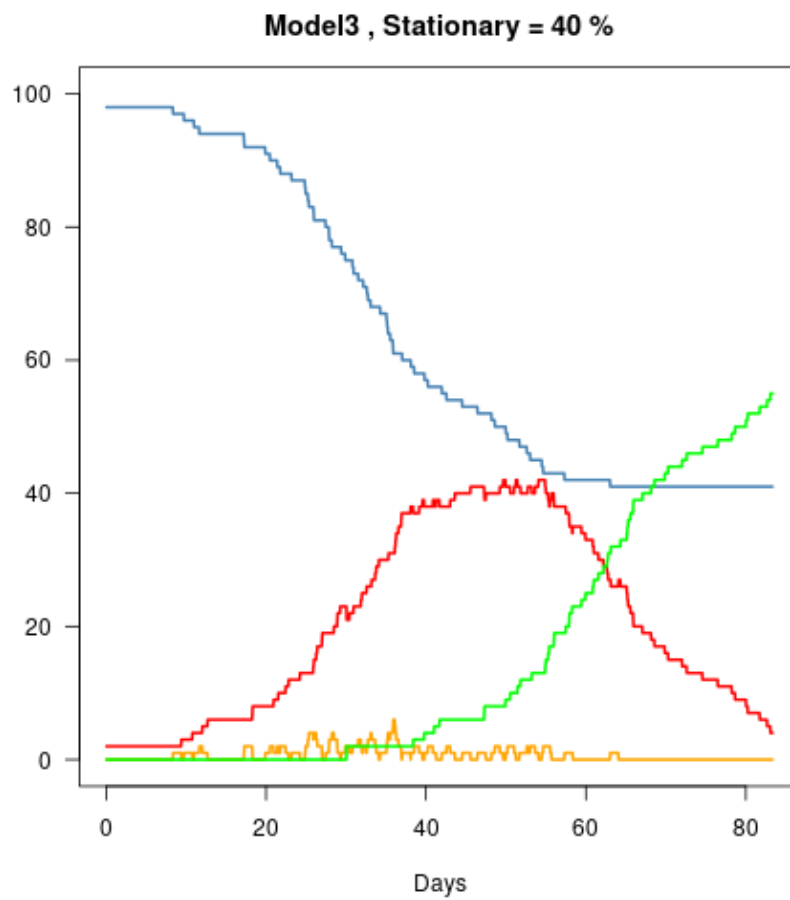


Figure 3: 40% shelter in place

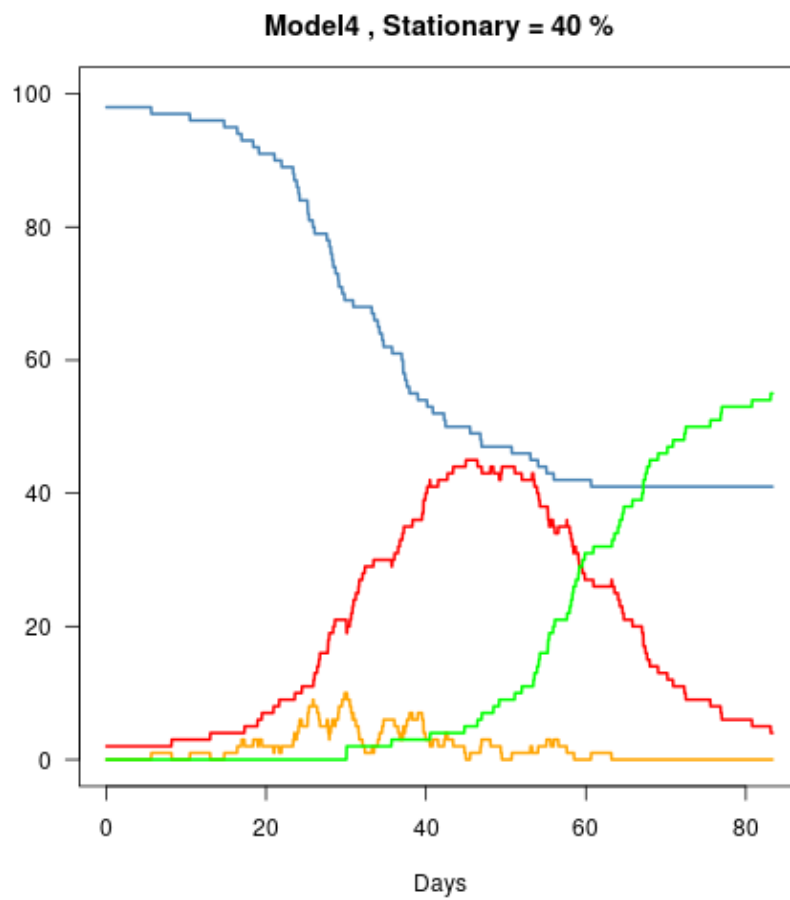


Figure 4: 40% shelter in place, exposed without symptoms.

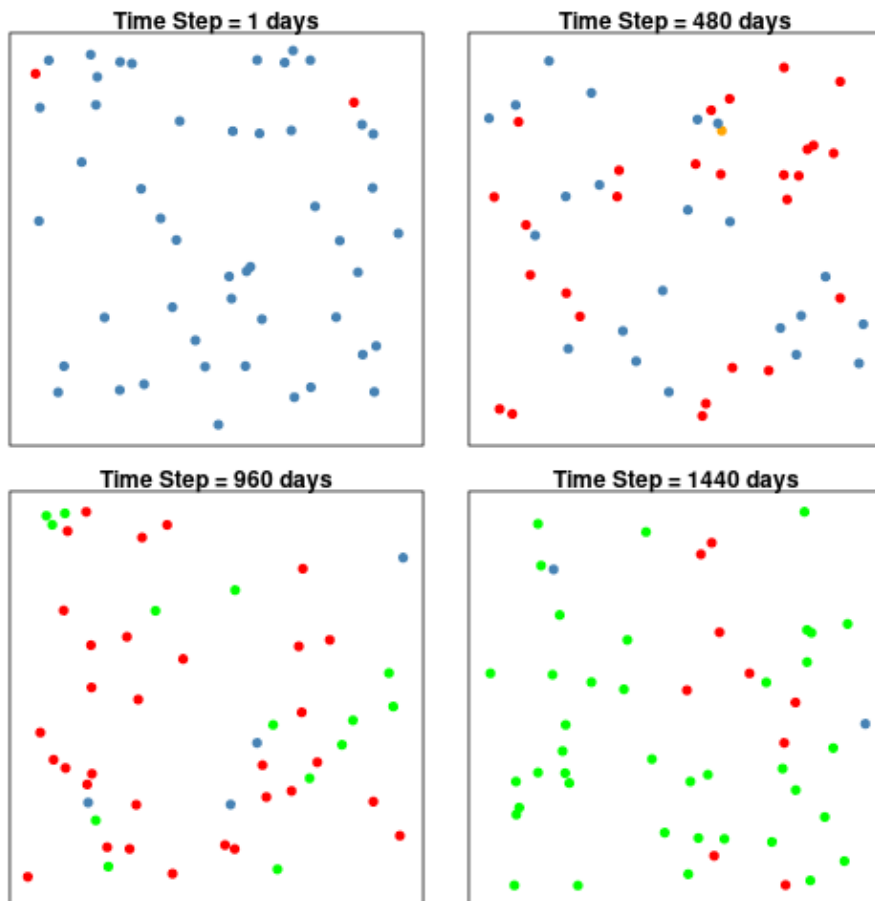


Figure 5: $N = 50$

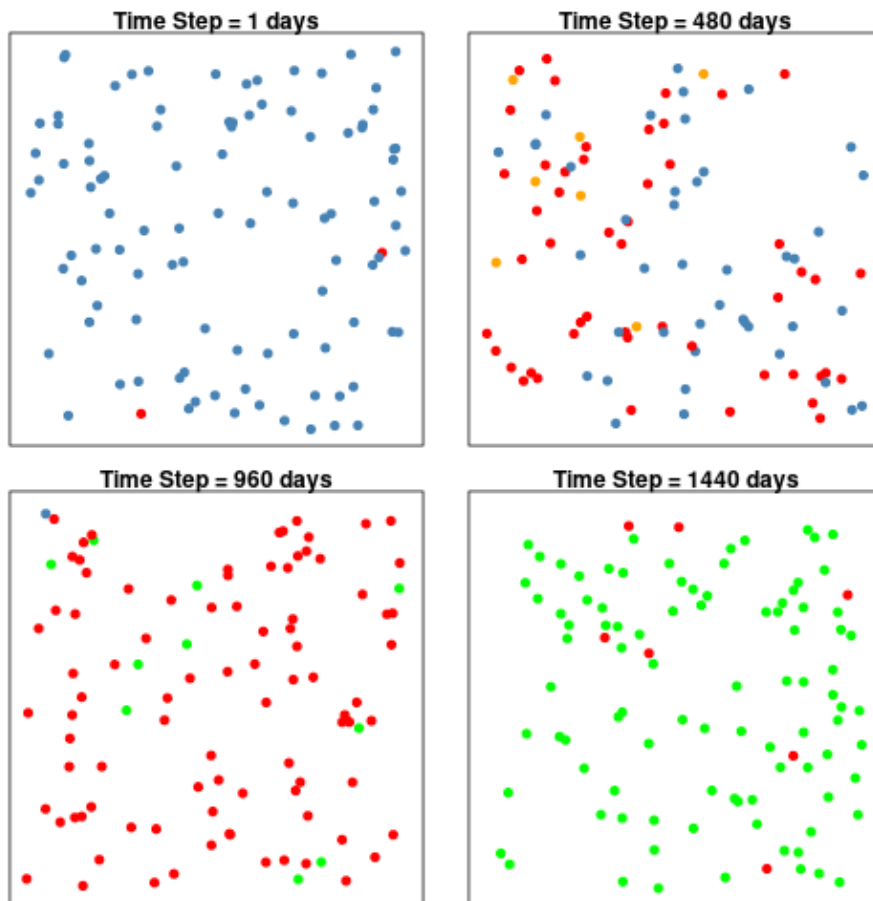


Figure 6: $N=100$

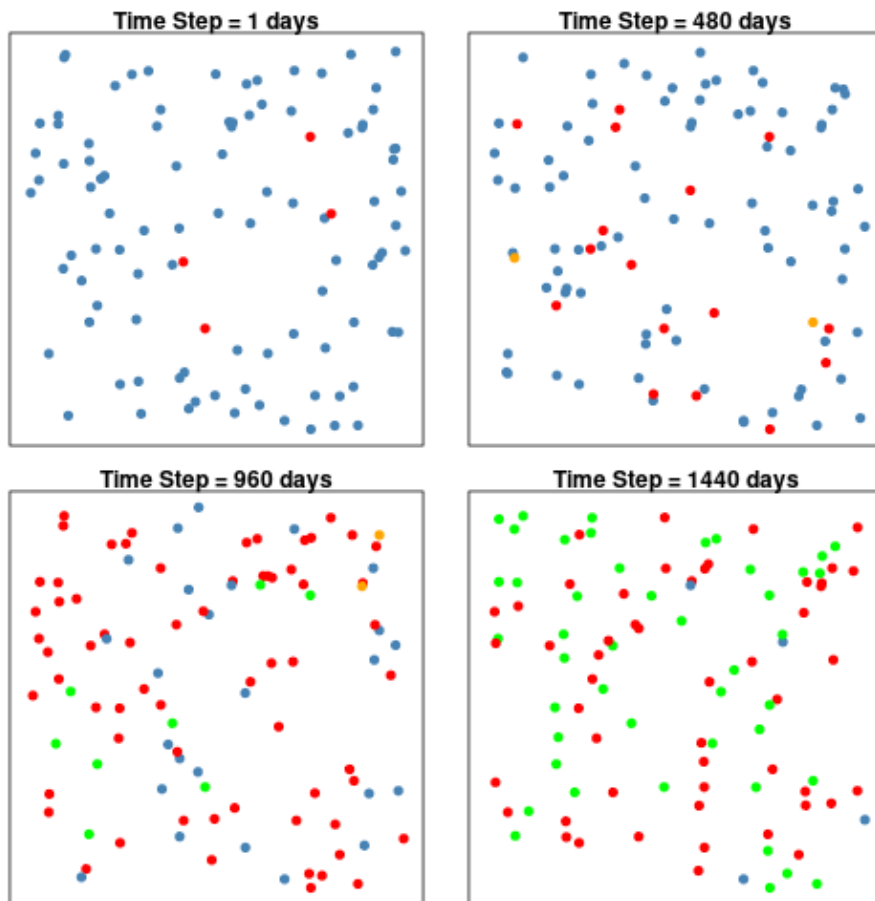


Figure 7: 40% shelter in place

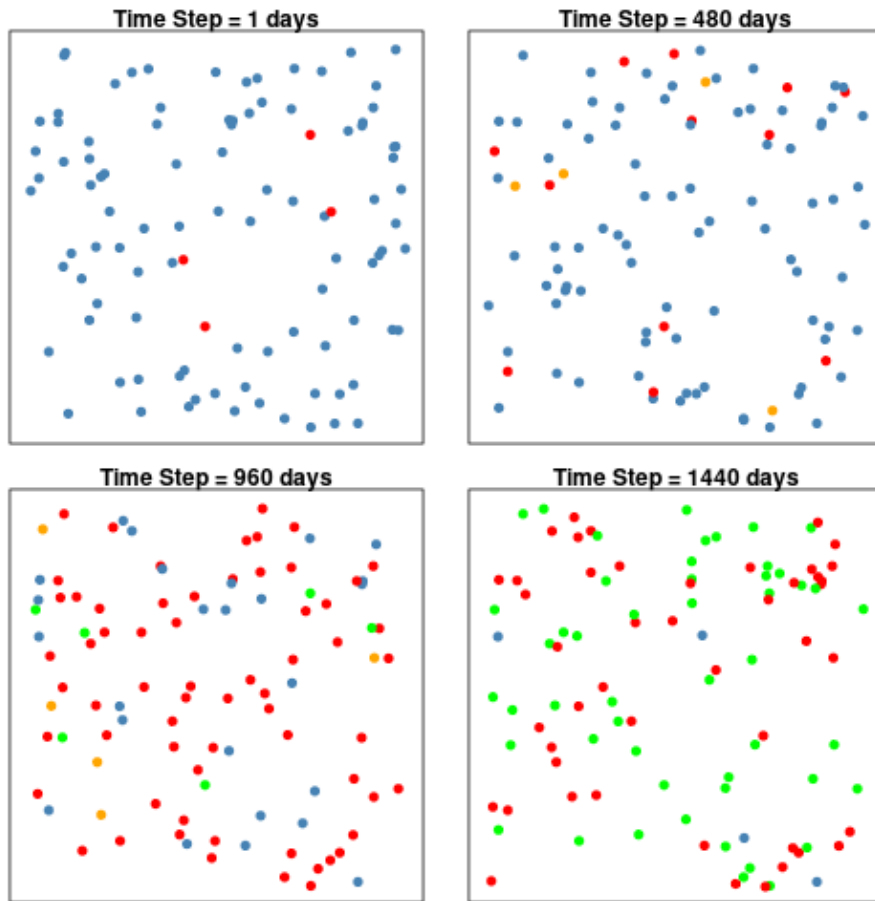


Figure 8: 40% shelter in place, exposed without symptoms.

3.3 Animating the Results

Still working on this... I have to export the file to my laptop then use photoshop to make a movie.

it would be nice to do this in r and then embedd in a pdf...

4 SEIRS with vital dynamics

You can also add vital dynamics to an SEIRS model, where μ and ν represent the birth and death rates, respectively. To maintain a constant population, assume that $\mu = \nu$. In steady state $\frac{dI}{dt} = 0$. The ODE then becomes:

$$\frac{dS}{dt} = \mu N - \frac{\beta SI}{N} + \xi R - \nu S \quad (14)$$

$$\frac{dE}{dt} = \frac{\beta SI}{N} - \sigma E - \nu E \quad (15)$$

$$\frac{dI}{dt} = \sigma E - \gamma I - \nu I \quad (16)$$

$$\frac{dR}{dt} = \gamma I - \xi R - \nu R \quad (17)$$

where $N = S + E + I + R$ is the total population.

5 Using Package EpiModel

dynamic, continuous-time, compartment models (DCMs),
stochastic, discrete-time, individual contact models (ICMs)
stochastic, discrete-time, network models
stochastic, discrete-time, individual contact models (ICMs)

```
control <- control.icm(type = "SIR", nsteps = 100, nsims = 10)
init <- init.icm(s.num = 997, i.num = 3, r.num = 0)

param <- param.icm(
  inf.prob = 0.05, act.rate = 10, rec.rate = 1/20,
  a.rate = (10.5/365)/1000, ds.rate = (7/365)/1000, di.rate = (14/365)/1000,
  dr.rate = (7/365)/1000)
sim <- icm(param, init, control)

sim

## EpiModel Simulation
## =====
## Model class: icm
##
```

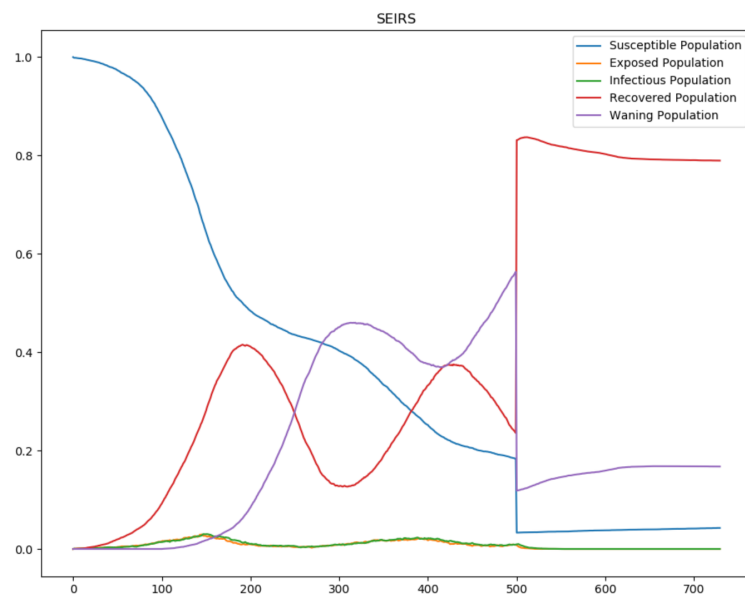
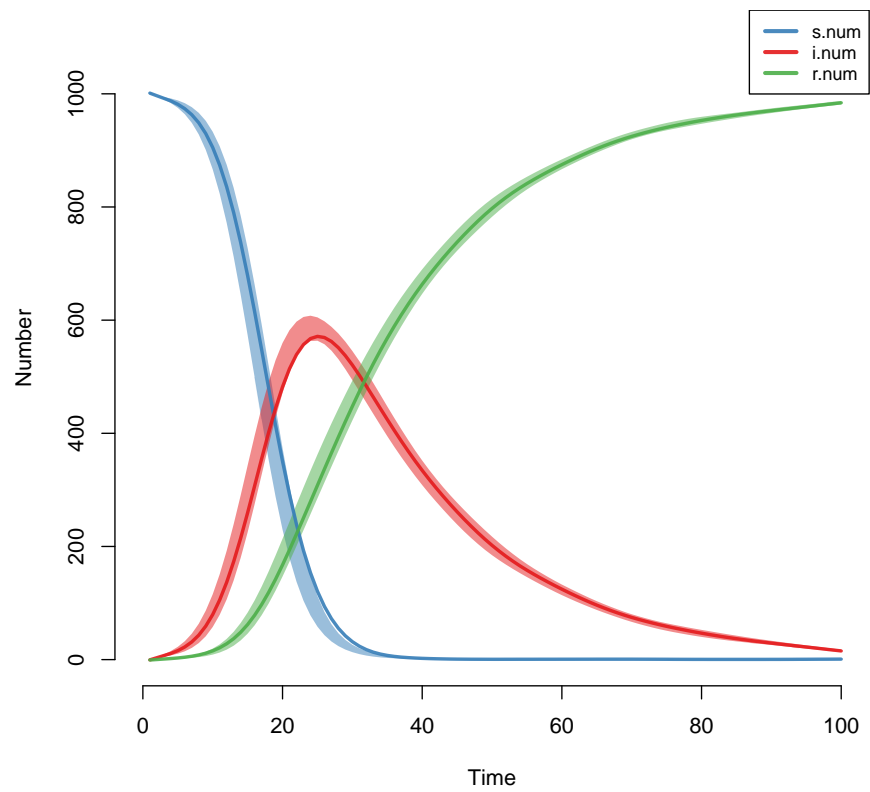


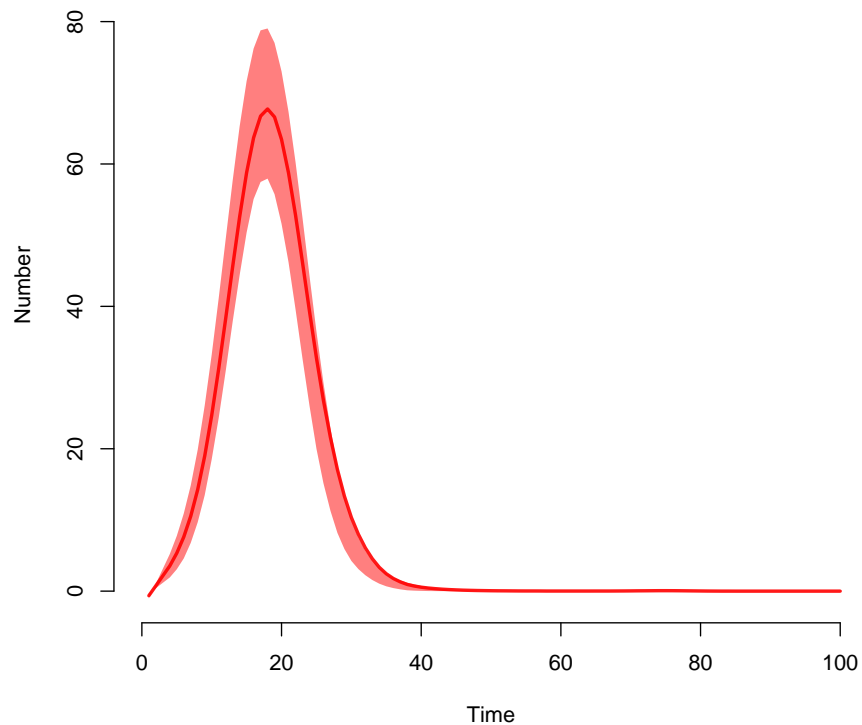
Figure 9: SEIRS

```
## Simulation Summary
## -----
## Model type: SIR
## No. simulations: 10
## No. time steps: 100
## No. groups: 1
##
## Model Parameters
## -----
## inf.prob = 0.05
## act.rate = 10
## rec.rate = 0.05
## a.rate = 2.876712e-05
## ds.rate = 1.917808e-05
## di.rate = 3.835616e-05
## dr.rate = 1.917808e-05
##
## Model Output
## -----
## Variables: s.num i.num num r.num si.flow ir.flow ds.flow
## di.flow dr.flow a.flow
```

```
plot(sim)
```



```
plot(sim, y = "si.flow", mean.col = "red", qnts.col = "red")
```



```
run_sir_sim <- function(inf_prob, act_rate, pop_size = 1000,
  i_num = 3, n_steps = 365, n_sims = 10, si_mean = 7.5, si_sd = 3.4) {

  # set up simulation parameters
  param <- param.icm(inf.prob = inf_prob, act.rate = act_rate,
    rec.rate = 1/20, a.rate = (10.5/365)/1000, ds.rate = (7/365)/1000,
    di.rate = (14/365)/1000, dr.rate = (7/365)/1000)
  init <- init.icm(s.num = pop_size - i_num, i.num = i_num,
    r.num = 0)
  control <- control.icm(type = "SIR", nsteps = n_steps, nsims = n_sims)

  # run the simulation
  sim <- icm(param, init, control)

  # collect the relevant results in a data frame
  incidence_rates <- as.data.frame(sim, out = "mean") %>% select(time,
```

```

    si.flow, i.num) %>% mutate(act_rate = act_rate, inf_prob = inf_prob,
                               total_cases = sum(si.flow), max_prev = max(i.num, na.rm = TRUE))

  # use the data frame of results to create an incidence()
  # object
  local_case_dates <- incidence_rates %>% filter(time <= 300,
    act.rate == act_rate, inf_prob == inf_prob) %>% select(time,
    si.flow) %>% uncount(si.flow) %>% pull(time)

  if (length(local_case_dates) > 0) {

    local_cases <- local_case_dates %>% incidence(.)

    # find the incidence peak from the incidence object
    peaky_blinder <- find_peak(local_cases)

    # recreate the incidence object using data only up to the
    # peak
    local_growth_phase_case_dates <- incidence_rates %>%
      filter(time <= peaky_blinder) %>% select(time, si.flow) %>%
      uncount(si.flow) %>% pull(time)

    local_growth_phase_cases <- local_growth_phase_case_dates %>%
      incidence(., last_date = peaky_blinder)

    # get a MLE estimate of the basic reproduction number, R0
    res <- get_R(local_growth_phase_cases, si_mean = si_mean,
      si_sd = si_sd)

    # add that as a column to the data frame of results
    incidence_rates <- incidence_rates %>% mutate(mle_R0 = res$R_ml)
  } else {
    # can't calculate R0 so just set to NA
    incidence_rates <- incidence_rates %>% mutate(mle_R0 = NA)
  }
  # return the data frame (which has just one row)
  return(incidence_rates)
} # end function definition

```

```

library(earlyR)
# set up an empty data frame to which to append results from
# each simulation
sims_incidence_rates <- tibble(time = integer(0), si.flow = numeric(0),
  i.num = numeric(0), act_rate = numeric(0), inf_prob = numeric(0),

```



```

    total_cases = numeric(0), max_prev = numeric(0), mle_R0 = numeric(0))

# the parameters to step through
act.rates <- c(10, 5, 2)
inf.probs <- c(0.05, 0.025, 0.01)

# loop through the parameter space
for (act.rate in act.rates) {
  for (inf.prob in inf.probs) {
    sims_incidence_rates <- sims_incidence_rates %>% bind_rows(run_sir_sim(inf.prob,
      act.rate))
  }
}

# create facet columns as descending ordered factors
sims_incidence_rates <- sims_incidence_rates %>% mutate(act_rate_facet_label = paste(act_rate,
  "exposures per day"), inf_prob_facet_label = paste("Probability of infection\nat each ex",
  inf_prob)) %>% arrange(desc(act_rate)) %>% mutate_at(vars(act_rate_facet_label),
  funs(factor(., levels = unique(.)))) %>% arrange(desc(inf_prob)) %>%
  mutate_at(vars(inf_prob_facet_label), funs(factor(., levels = unique(.)))) %>%
  arrange(desc(act_rate), desc(inf_prob), time)

## Warning: funs() is soft deprecated as of dplyr 0.8.0
## Please use a list of either functions or lambdas:
##
## # Simple named list:
## list(mean = mean, median = median)
##
## # Auto named with 'tibble::lst()':
## tibble::lst(mean, median)
##
## # Using lambdas
## list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
## This warning is displayed once per session.

# add annotation text for each facet
sims_incidence_rates_facet_annotations <- sims_incidence_rates %>%
  mutate(label = paste("R0 =", format(mle_R0, digits = 3),
    "\n", round(100 * total_cases/1000, digits = 0), "% of population infected")) %>%
  select(inf_prob_facet_label, act_rate_facet_label, label) %>%
  distinct()

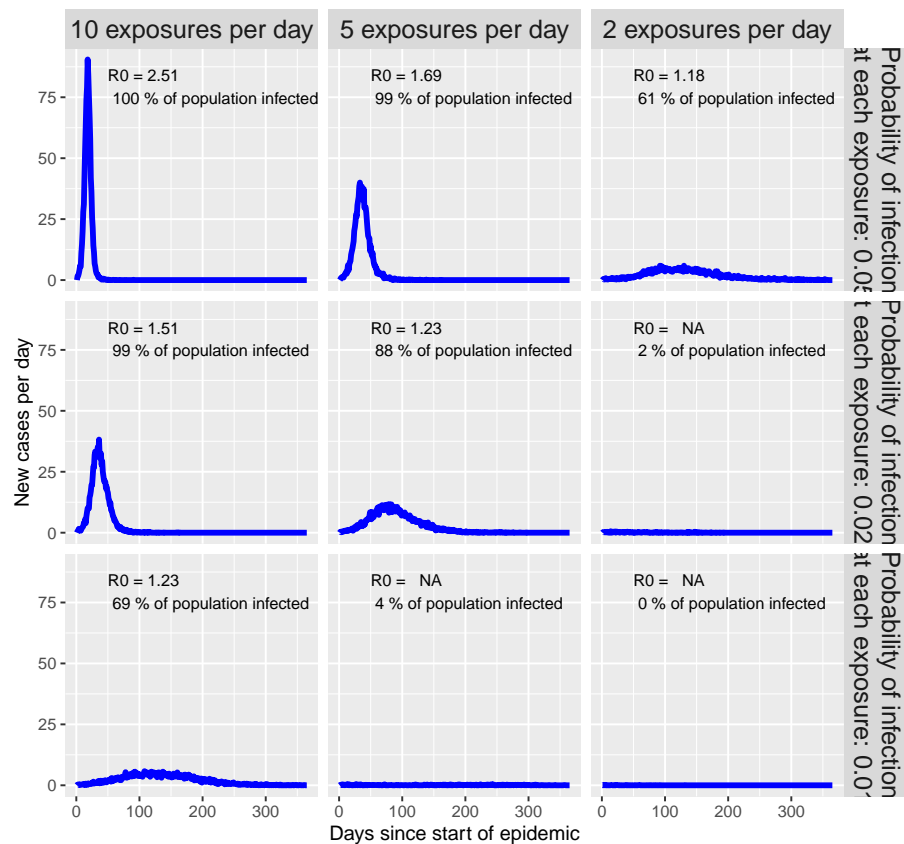
sims_incidence_rates %>% filter(time <= 365) %>% ggplot(aes(x = time,
  y = si.flow)) + geom_line(colour = "blue", size = 1.5) +
  facet_grid(inf_prob_facet_label ~ act_rate_facet_label) +

```

```
geom_text(data = sims_incidence_rates_facet_annotations,
  mapping = aes(x = 50, y = 0.8 * max(sims_incidence_rates$si.flow,
    na.rm = TRUE), label = label), parse = FALSE, hjust = 0,
  vjust = 0, size = 3) + labs(x = "Days since start of epidemic",
  y = "New cases per day", title = "Modelling of new cases of COVID-19 per day: incidence
  subtitle = paste("with varying levels of social mixing (exposures per day)",
    "and probabilities of infection at each exposure")) +
  theme(legend.position = "top", strip.text = element_text(size = 14))
```

Modelling of new cases of COVID-19 per day: incidence rate

with varying levels of social mixing (exposures per day) and probabilities of infection at each expo



6 SEIQHRF –Wow, now we can model stuff!

```
tic("Time to complete")
```

```

source_files <- c("_icm.mod.init.seiqhrf.R", "_icm.mod.status.seiqhrf.R",
  "_icm.mod.vital.seiqhrf.R", "_icm.control.seiqhrf.R", "_icm.utils.seiqhrf.R",
  "_icm.saveout.seiqhrf.R", "_icm.icm.seiqhrf.R")

src_path <- paste0("./_posts/2020-03-18-modelling-the-effects-of-public-health-",
  "interventions-on-covid-19-transmission-part-2/")

gist_url <- "https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7"

local_source <- FALSE

for (source_file in source_files) {
  if (local_source) {
    source(paste(src_path, source_file, sep = ""))
  } else {
    source_gist(gist_url, filename = source_file)
  }
}

## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is 323250b6e71cd44c3d201fe2b7c85df896a65368
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is 63756c53135ded7eb697831f59aa24b84edda344
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is e864b3c4043b8ec22ed65408bfe36d7b75a3afaa
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is 8514ecf3080ca0b96d53d3d81e0019af4cdd1c07
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is cfd032f1285332f8c4d2cc1a13c8c92920ae9370
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is 14b18703fc9e46d31e279a364df4e89369ddae98
## Sourcing https://gist.github.com/timchurches/92073d0ea75cfbd387f91f7c6e624bd7
## SHA-1 hash of file is 182364e711588462e32ff3254947027f78c920ed

# function to set-up and run the baseline simulations
simulate <- function(# control.icm params
  type = "SEIQHRF",
  nsteps = 366,
  nsims = 8,
  ncores = 4,
  prog.rand = FALSE,
  rec.rand = FALSE,
  fat.rand = TRUE,
  quar.rand = FALSE,
  hosp.rand = FALSE,

```

```

disch.rand = TRUE,
infection.FUN = infection.seiqhrf.icm,
recovery.FUN = progress.seiqhrf.icm,
departures.FUN = departures.seiqhrf.icm,
arrivals.FUN = arrivals.icm,
get_prev.FUN = get_prev.seiqhrf.icm,
# init.icm params
s.num = 9997,
e.num=0,
i.num = 3,
q.num=0,
h.num=0,
r.num = 0,
f.num = 0,
# param.icm params
inf.prob.e = 0.02,
act.rate.e = 10,
inf.prob.i = 0.05,
act.rate.i = 10,
inf.prob.q = 0.02,
act.rate.q = 2.5,
quar.rate = 1/30,
hosp.rate = 1/100,
disch.rate = 1/15,
prog.rate = 1/10,
prog.dist.scale = 5,
prog.dist.shape = 1.5,
rec.rate = 1/20,
rec.dist.scale = 35,
rec.dist.shape = 1.5,
fat.rate.base = 1/50,
hosp.cap = 40,
fat.rate.overcap = 1/25,
fat.tcoeff = 0.5,
vital = TRUE,
a.rate = (10.5/365)/1000,
a.prop.e = 0.01,
a.prop.i = 0.001,
a.prop.q = 0.01,
ds.rate = (7/365)/1000,
de.rate = (7/365)/1000,
di.rate = (7/365)/1000,
dq.rate = (7/365)/1000,
dh.rate = (20/365)/1000,
dr.rate = (7/365)/1000,

```

```

        out="mean"
    ) {

control <- control.icm(type = type,
                      nsteps = nsteps,
                      nsims = nsims,
                      ncores = ncores,
                      prog.rand = prog.rand,
                      rec.rand = rec.rand,
                      infection.FUN = infection.FUN,
                      recovery.FUN = recovery.FUN,
                      arrivals.FUN = arrivals.FUN,
                      departures.FUN = departures.FUN,
                      get_prev.FUN = get_prev.FUN)

init <- init.icm(s.num = s.num,
                 e.num = e.num,
                 i.num = i.num,
                 q.num = q.num,
                 h.num = h.num,
                 r.num = r.num,
                 f.num = f.num)

param <- param.icm(inf.prob.e = inf.prob.e,
                   act.rate.e = act.rate.e,
                   inf.prob.i = inf.prob.i,
                   act.rate.i = act.rate.i,
                   inf.prob.q = inf.prob.q,
                   act.rate.q = act.rate.q,
                   quar.rate = quar.rate,
                   hosp.rate = hosp.rate,
                   disch.rate = disch.rate,
                   prog.rate = prog.rate,
                   prog.dist.scale = prog.dist.scale,
                   prog.dist.shape = prog.dist.shape,
                   rec.rate = rec.rate,
                   rec.dist.scale = rec.dist.scale,
                   rec.dist.shape = rec.dist.shape,
                   fat.rate.base = fat.rate.base,
                   hosp.cap = hosp.cap,
                   fat.rate.overcap = fat.rate.overcap,
                   fat.tcoeff = fat.tcoeff,
                   vital = vital,
                   a.rate = a.rate,
                   a.prop.e = a.prop.e,

```

```

        a.prop.i = a.prop.i,
        a.prop.q = a.prop.q,
        ds.rate = ds.rate,
        de.rate = de.rate,
        di.rate = di.rate,
        dq.rate = dq.rate,
        dh.rate = dh.rate,
        dr.rate = dr.rate)

sim <- icm.seiqhrf(param, init, control)
sim_df <- as.data.frame(sim, out=out)

return(list(sim=sim, df=sim_df))
}

```

```

# define a function to extract timings and assemble a data
# frame
get_times <- function(simulate_results) {

  sim <- simulate_results$sim

  for (s in 1:sim$control$nsims) {
    if (s == 1) {
      times <- sim$times[[paste("sim", s, sep = "")]]
      times <- times %>% mutate(s = s)
    } else {
      times <- times %>% bind_rows(sim$times[[paste("sim",
        s, sep = "")]] %>% mutate(s = s))
    }
  }

  times <- times %>% mutate(infTime = ifelse(infTime < 0, -5,
    infTime), expTime = ifelse(expTime < 0, -5, expTime)) %>%
    mutate(incubation_period = infTime - expTime, illness_duration = recovTime -
      expTime, illness_duration_hosp = dischTime - expTime,
      hosp_los = dischTime - hospTime, quarantine_delay = quarTime -
        infTime, survival_time = fatTime - infTime) %>%
    select(s, incubation_period, quarantine_delay, illness_duration,
      illness_duration_hosp, hosp_los, survival_time) %>%
    pivot_longer(-s, names_to = "period_type", values_to = "duration") %>%
    mutate(period_type = factor(period_type, levels = c("incubation_period",
      "quarantine_delay", "illness_duration", "illness_duration_hosp",
      "hosp_los", "survival_time"), labels = c("Incubation period",
      "Delay entering isolation", "Illness duration", "Illness duration (hosp)",

```

```

      "Hospital care required duration", "Survival time of case fatalities"),
      ordered = TRUE))
    return(times)
  }

```

```
times <- get_times(baseline_sim)
```

```
## Error in get_times(baseline_sim): object 'baseline_sim' not found
```

```
times %>% filter(duration <= 30) %>% ggplot(aes(x = duration)) +
  geom_bar() + facet_grid(period_type ~ ., scales = "free_y") +
  labs(title = "Duration frequency distributions", subtitle = "Baseline simulation")

```

```
## Error in UseMethod("filter"): no applicable method for 'filter_'
applied to an object of class "c('double', 'numeric')"
```

```
baseline_sim <- simulate(ncores = 4)
```

```
baseline_plot_df <- baseline_sim$df %>% # use only the prevalence columns
select(time, s.num, e.num, i.num, q.num, h.num, r.num, f.num) %>%
  # examine only the first 100 days since it is all over by
  # then using the default parameters
filter(time <= 100) %>% pivot_longer(-c(time), names_to = "compartment",
  values_to = "count")

```

```

# define a standard set of colours to represent compartments
compcols <- c(s.num = "yellow", e.num = "orange", i.num = "red",
  q.num = "cyan", h.num = "magenta", r.num = "lightgreen",
  f.num = "black")

```

```

complabels <- c(s.num = "Susceptible", e.num = "Infected/asymptomatic",
  i.num = "Infected/infectious", q.num = "Self-isolated", h.num = "Requires hospitalisation",
  r.num = "Recovered", f.num = "Case fatality")

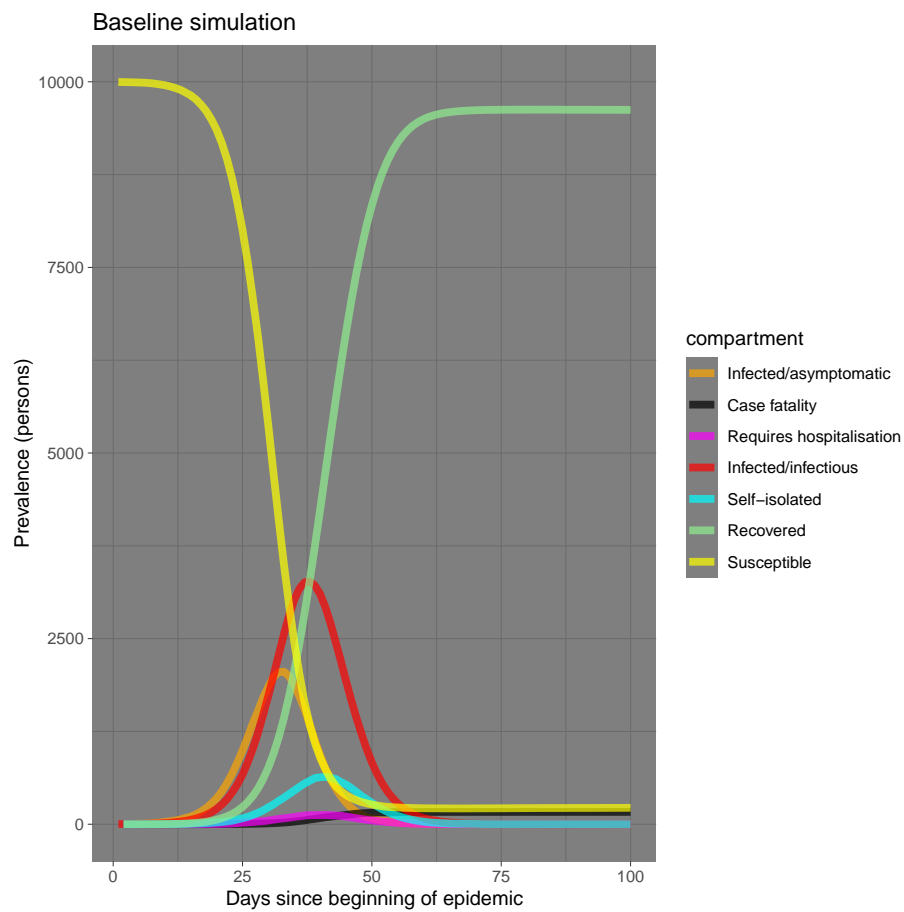
```

```

baseline_plot_df %>% ggplot(aes(x = time, y = count, colour = compartment)) +
  geom_line(size = 2, alpha = 0.7) + scale_colour_manual(values = compcols,
  labels = complabels) + theme_dark() + labs(title = "Baseline simulation",
  x = "Days since beginning of epidemic", y = "Prevalence (persons)")

```

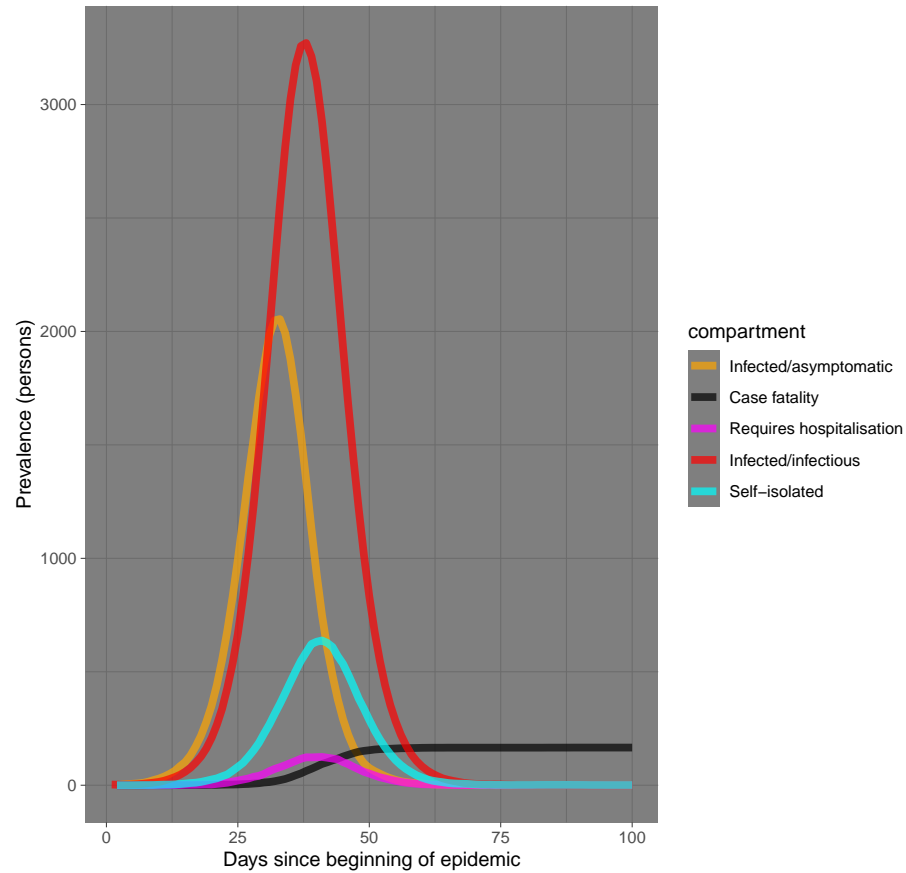
```
## Warning: Removed 5 row(s) containing missing values (geom_path).
```



```
baseline_plot_df %>% filter(compartment %in% c("e.num", "i.num",
  "q.num", "h.num", "f.num")) %>% ggplot(aes(x = time, y = count,
  colour = compartment)) + geom_line(size = 2, alpha = 0.7) +
  scale_colour_manual(values = compcols, labels = complabels) +
  theme_dark() + labs(title = "Baseline simulation", x = "Days since beginning of epidemic",
  y = "Prevalence (persons)")

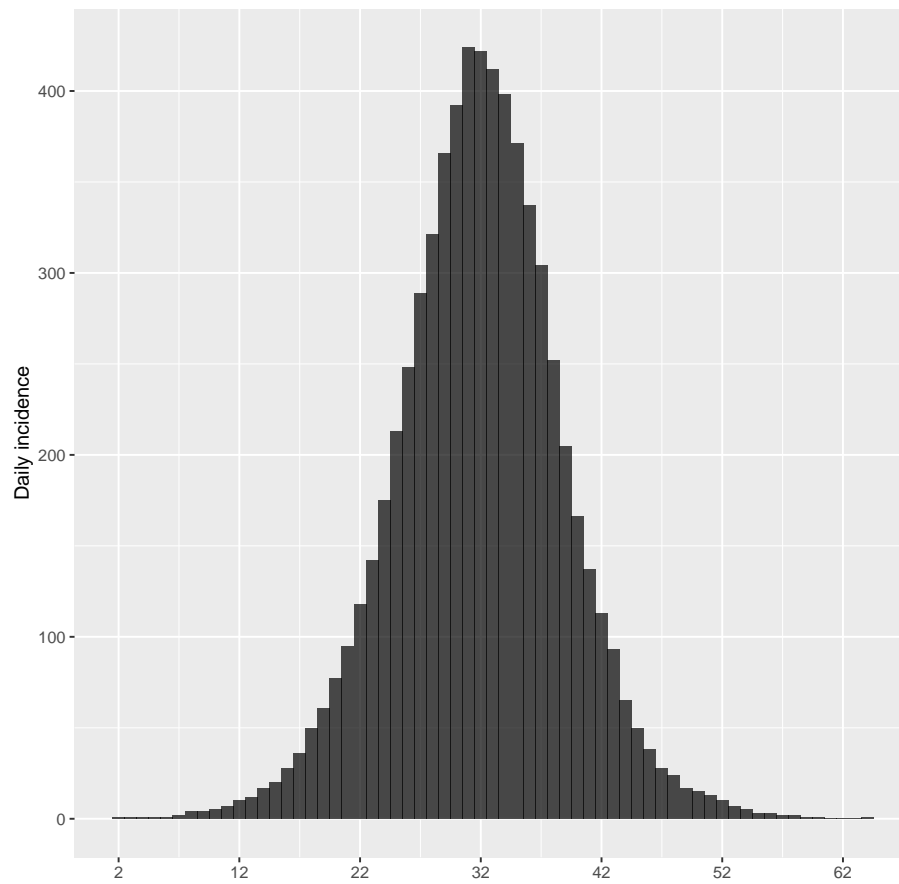
## Warning: Removed 4 row(s) containing missing values (geom_path).
```


Baseline simulation



```
# get the S-> E compartment flow, which is our daily
# incidence rate
incidence_counts <- baseline_sim$df %>% select(time, se.flow)
# uncount them
incident_case_dates <- incidence_counts %>% uncount(se.flow) %>%
  pull(time)
# convert to an incidence object
incidence_all <- incident_case_dates %>% incidence(.)

# plot the incidence curve
plot(incidence_all)
```



```
toc()
```

```
## Time to complete: 38.996 sec elapsed
```