SIR_model

September 12, 2023

0.0.1 The SIR model of disease spread SOLUTIONS

S is the number of susceptible individuals.

I is the number of infected individuals.

R is the number of resistant/recovered individuals.

All three variables change over time and we can understand how by thinking about their *rates of change* (pillar 2).

Susceptible individuals become infected when they come into contact with an infected individual and the disease is transmitted. The number of such contacts is (roughly) **SR**, but not every contact transmits the disease, so we multiply by a (positive) constant **a** that measures how contagious the disease is. This gives **S'= -aSR**.

An individual becomes resistant by recovering from the infection. People recover at a constant rate **b**, so at every time step we get **bI** newly recovered individuals. This gives **R' = bI**.

The number of infected individuals is just the combination of the two things we've already done. We add the newly infected and subtract the newly recovered. This gives **I' = aSI - bI**.

Putting that all together, we have:

S'= -aSR

I' = aSI - bI

R' = bI

What we have set up is a *system of differential equations*, which we can deal with using pillar 5: one step at a time. To go one step forward in time we simply add the changes based on our current values:

S(t+1) = S(t) - aS(t)I(t)

I(t+1) = I(t) + aS(t)I(t) - bI(t)

R(t+1) = R(t) + bI(t)

It should have been apparent in problem 3 of the worksheet that doing the calculations by hand gets tedious fairly quickly. Instead, let's have a computer do all the work:

[7]:
```
# Set values for our variables and constants (this is a comment; Python ignores␣
↪it):
S0=100
```

```
I0=1
R0=0
a=0.01
b=0.125

# Now have the computer do the math:
S1=S0-a*S0*I0
I1=I0+a*S0*I0-b*I0
R1=R0+b*I0

# Output the results:
print('S(1)=', S1)
print('I(1)=', I1)
print('R(1)=', R1)

# Press shift+enter to run the python code
```

```
S(1)= 99.0
I(1)= 1.875
R(1)= 0.125
```

0.1 Problem 1:

Copy, paste, and modify the code above to generate estimates for **S(2), I(2), R(2)** and **S(3), I(3), R(3)**. Put your code in the cell below. Remember shift+enter to run the code.

[9]:
```
# Now have the computer do the math:
S2=S1-a*S1*I1
I2=I1+a*S1*I1-b*I1
R2=R1+b*I1

# Output the results:
print('S(2)=', S2)
print('I(2)=', I2)
print('R(2)=', R2)

# Now have the computer do the math:
S3=S2-a*S2*I2
I3=I2+a*S2*I2-b*I2
R3=R2+b*I2

# Output the results:
print('S(3)=', S3)
print('I(3)=', I3)
print('R(3)=', R3)
```

```
S(2)= 97.14375
I(2)= 3.496875
```

```
R(2)= 0.359375
S(3)= 93.74675449218749
I(3)= 6.4567611328125
R(3)= 0.796484375
```

We're still not fully taking advantage of the computer. We'd really like to make the computer *iterate* without our having to copy, paste, and modify. We can do that with a *for loop* (or any other kind of loop or recursive structure; I'd like to use a *for loop* later, so I'm introducing it here).

Python still knows everything we've coded, even things from earlier cells.

[15]:
```
# Set a number of days
d=30

# Start keeping track of our values as lists
S=[S0]
I=[I0]
R=[R0]

# Do the math for that many steps and append the new numbers to the ends of the␣
 ↪lists
for t in range(d):
    S.append(S[t]-a*S[t]*I[t])
    I.append(I[t]+a*S[t]*I[t]-b*I[t])
    R.append(R[t]+b*I[t])

# Output the results:
print('S(7)=', S[7])
print('I(7)=', I[7])
print('R(7)=', R[7])

print('S(30)=', S[30])
print('I(30)=', I[30])
print('R(30)=', R[30])
```

```
S(7)= 40.74099015505391
I(7)= 50.402956478033246
R(7)= 9.856053366912828
S(30)= 0.0030111763239796478
I(30)= 4.751003952081127
R(30)= 96.24598487159484
```

0.2 Problem 2:

Modify the code above to get estimates for **S, I,** and **R** at **d=7** and **d=30**.

0.2.1 Answers above

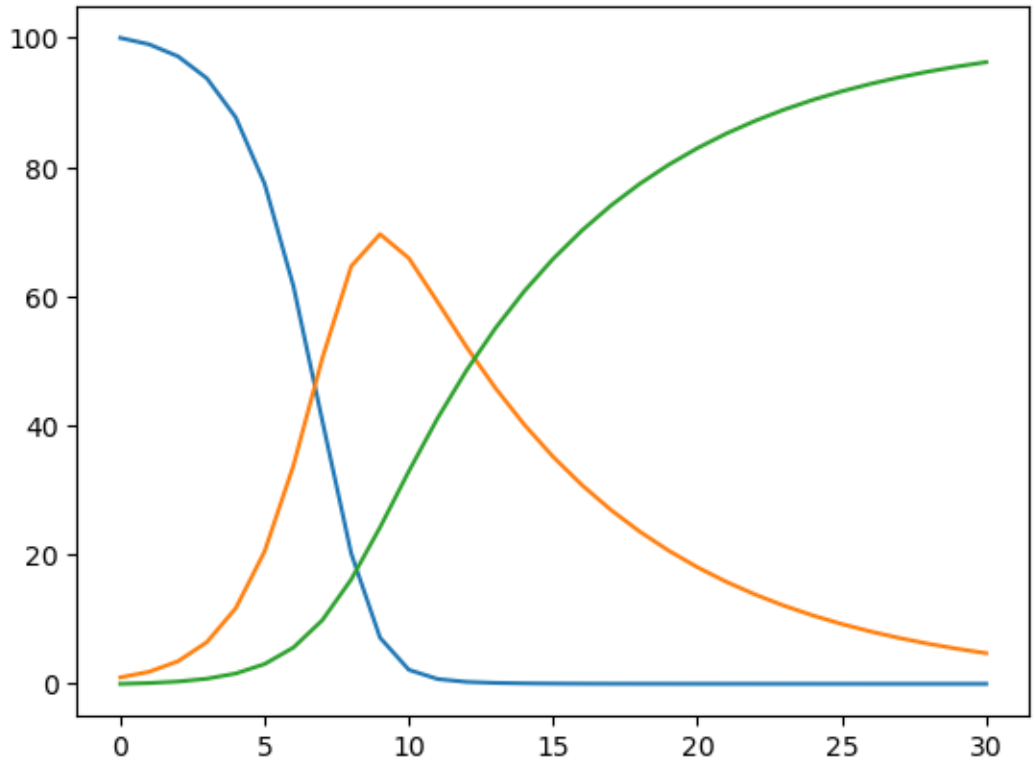
0.3 Problem 3:

Now that we have lists of values for **S, I,** and **R**, we can also start to make plots. Copy, paste, and modify the code below to generate plots for **I** and **R**. Are they what you expected to see?

[20]:
```
#Tell python that we want to use a plotting function
import matplotlib.pyplot as plt

# Generate a list of x-values to go with the y-values we already have
x=list(range(d))
x.append(d)

# Generate the plot for S and display it
plt.plot(x, S) # 30 day prediction, one step per day
plt.plot(x, I)
plt.plot(x, R)
plt.show()
```

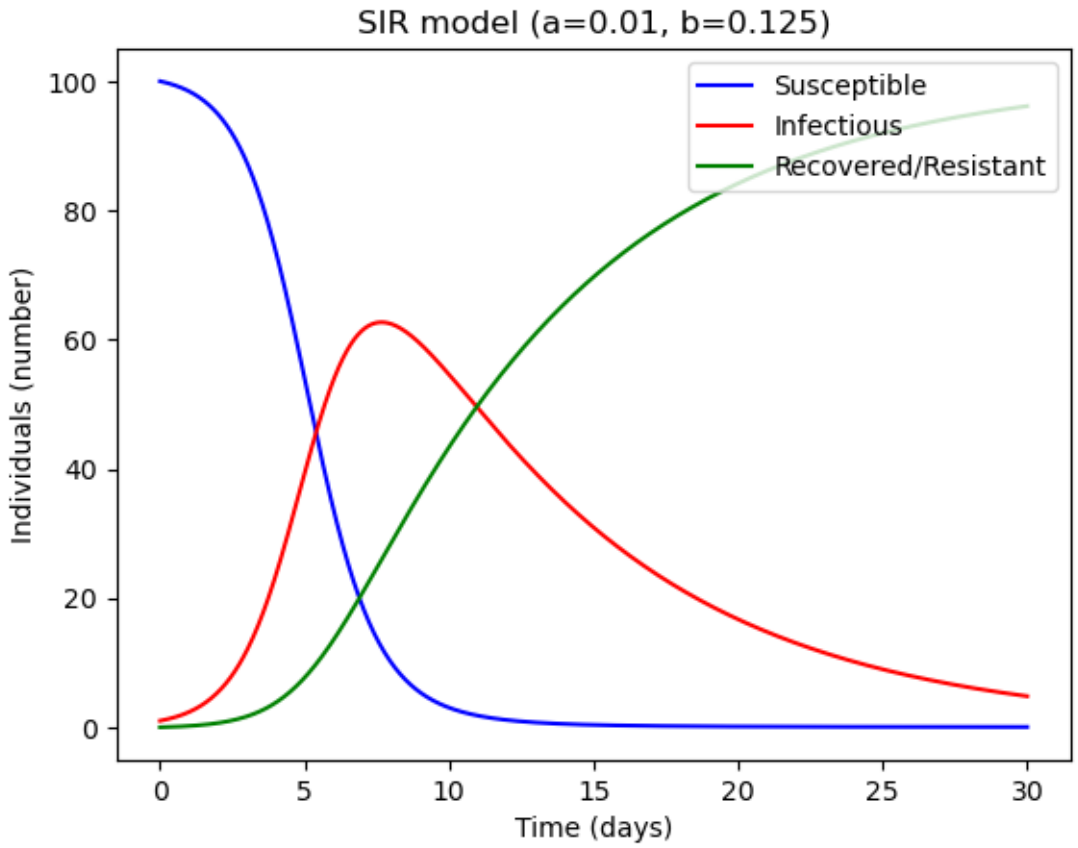


We can collect all of this into a single little program called sir_model(a, b). This program also allows for steps smaller than one day at a time. To take smaller steps, we simply scale **a** and **b**: dividing by **n** gives the values for **n** steps per day. For example, if **b=0.125** per day, then we divide by 2 to get **0.0625** per 12 hours; that's 2 steps per day.

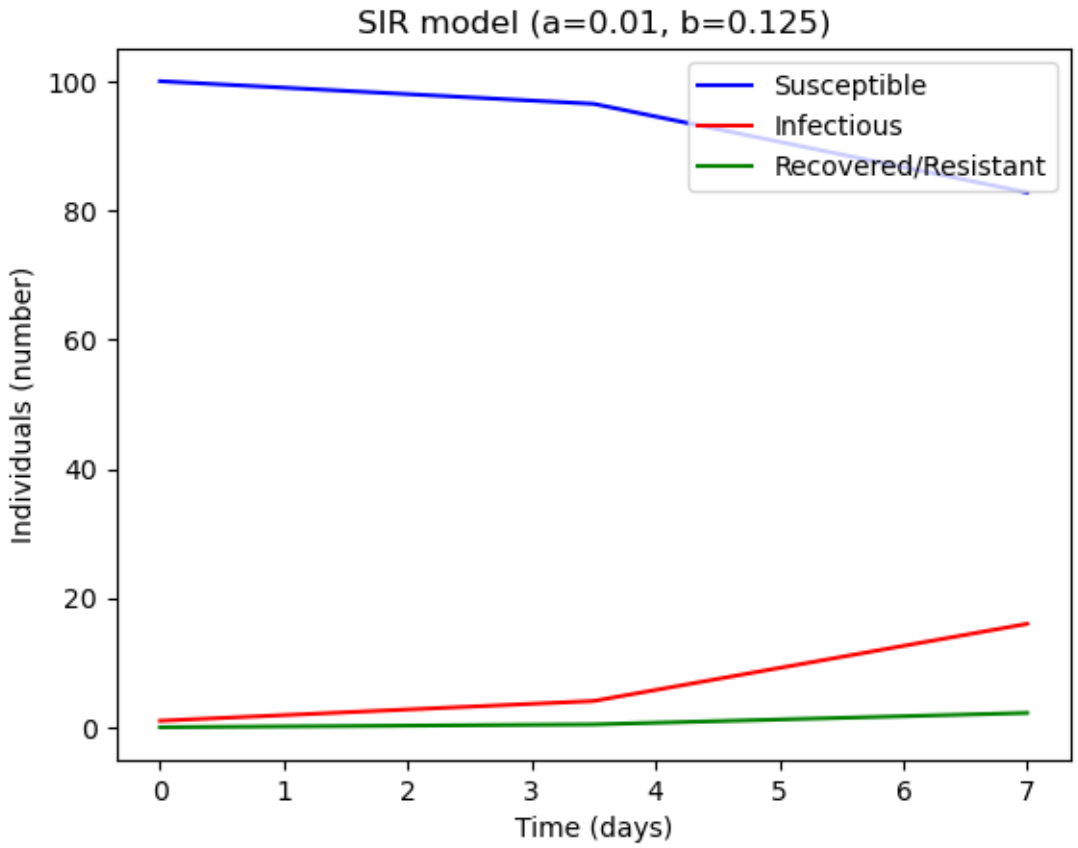
You can read (and change) the code below, if you want. Otherwise, skip to the examples below. Run the code to see what we're getting.

[22]:
```
def sir_model(a, b, S0=100, I0=1, R0=0, days=30, n=1000):
    # Input variables a and b are our infection and recovery rate parameters. Should be measured using days as the unit of time.
    # Inputs S0, R0, and I0 are the initial values for S, I, and R. Defaults are 100, 1, and 0, respectively.
    # Input days is the number of days we want to predict. Default is 7 days (one week).
    # Input n is the number of steps to take in making our prediction. Default is 1000.
    # In use, a and b will be scaled by multiplication by days/n
    scale=days/n # set our scaling factor
    d=[0] # a list of times mesured in days (the x values for our plot)
    s=[S0] # a list of values for S at the times in d (y values for the plot of S)
    i=[I0] # a list of values for I at the times in d (y values for the plot of I)
    r=[R0] # a list of values for R at the times in d (y values for the plot of R)
    for x in range(n): # this loop steps through the days, calculating S(t+1), I(t+1), and R(t+1)
        d.append((x+1)*scale)
        s.append(s[x]-(a*scale*s[x]*i[x]))
        i.append(i[x]+a*scale*s[x]*i[x]-b*scale*i[x])
        r.append(r[x]+b*scale*i[x])
    # Now we just need to plot our results
    plt.plot(d,s, color='blue', label='Susceptible')
    plt.plot(d,i, color='red', label='Infectious')
    plt.plot(d,r, color='green', label='Recovered/Resistant')
    plt.xlabel('Time (days)')
    plt.ylabel('Individuals (number)')
    plt.title('SIR model (a='+str(a)+', b='+str(b)+')')
    plt.legend(loc='upper right')
    plt.show()
```

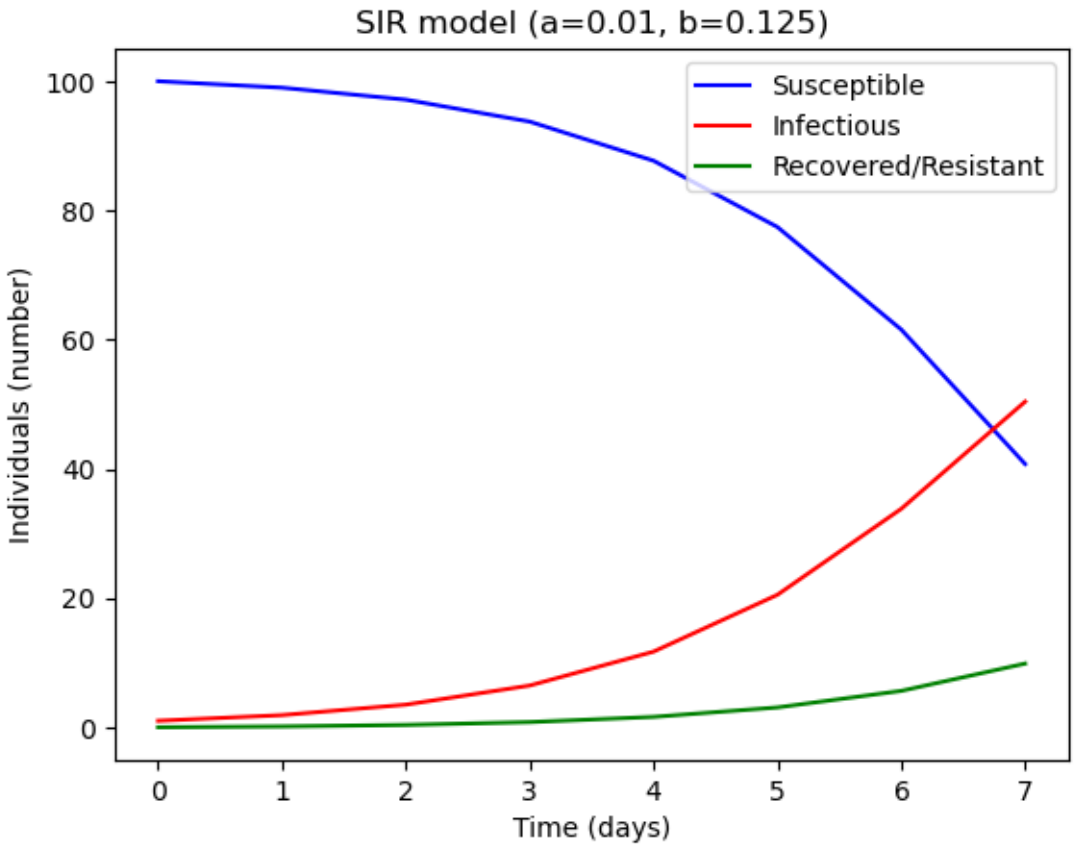
[24]:
```
sir_model(0.01, 0.125, 100, 1, 0, 30, 1000) # model a fast-spreading disease with 8 day recovery. 1 week projection in 1000 steps.
```



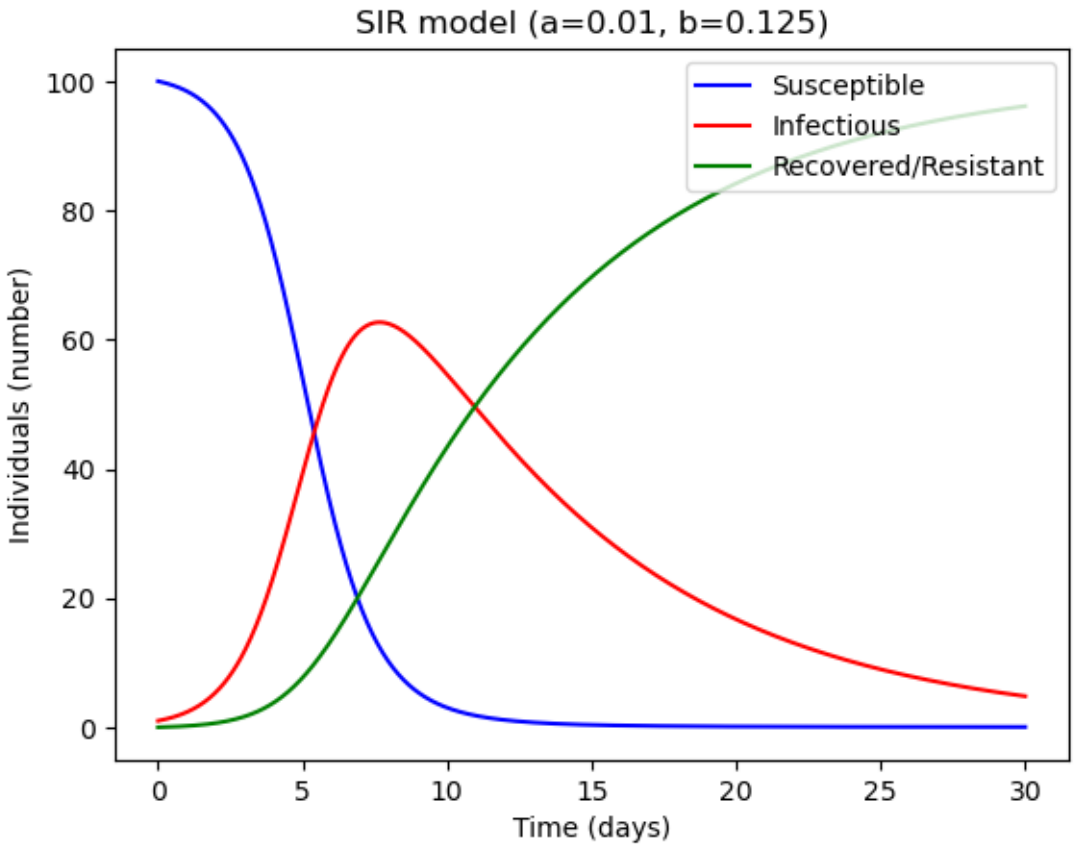
[26]:
```
sir_model(0.01, 0.125, 100, 1, 0, 7, 2) # same disease, but projecting 1 week␣
↪forward in just 2 steps
```



[28]:
```
sir_model(0.01, 0.125, 100, 1, 0, 7, 7) # now 7 steps in a week; one step per␣
↪day
```



[30]:
```
sir_model(0.01, 0.125, 100, 1, 0, 30, 1000) # same diesase projected 30 days in␣
↪1000 steps
```



0.3.1 Problem 4

Try modifying the values of **a** and **b** (one at a time, then both) to answer the following. Include things like peak values for **I** and a general estimate of how long the disease affects a significant portion of the population. You may also need to adjust the time scale to get good estimates.

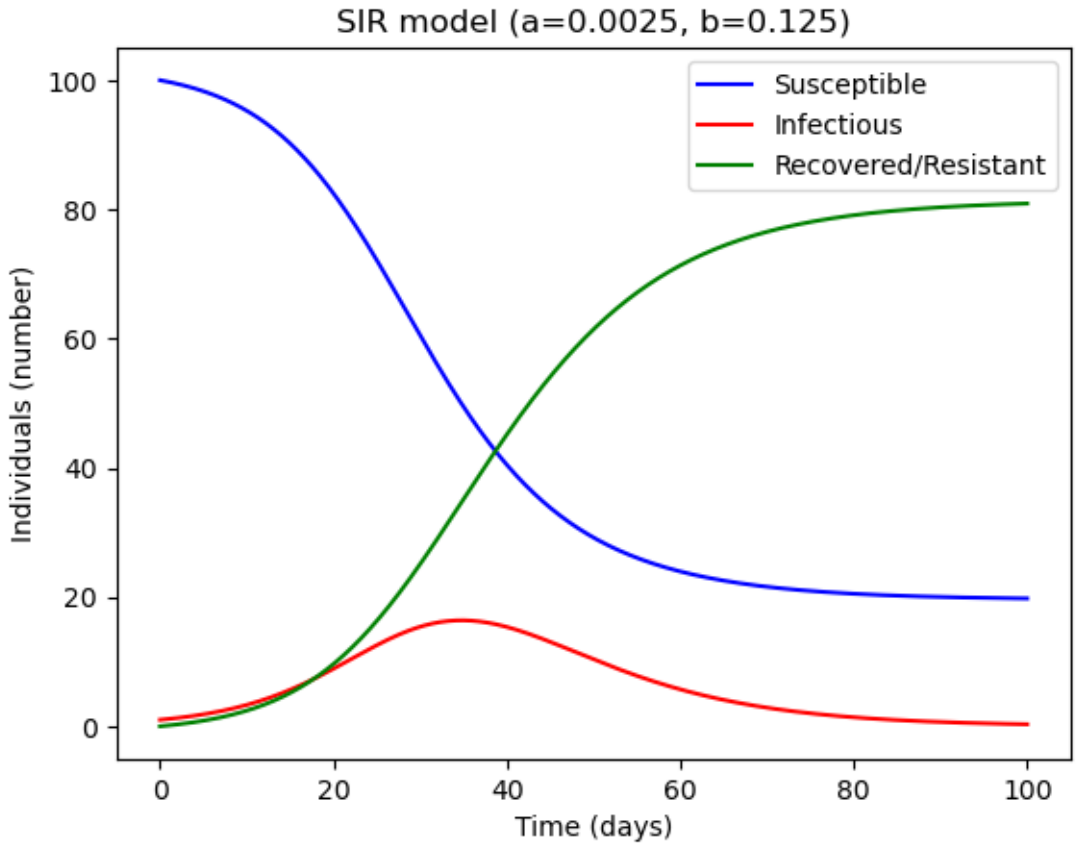
- How does **a** affect projections?
- How does **b** affect projections?
- Are there values of **a** and **b** that keep some people from ever getting sick?
- What kinds of things in the real world affect **a** and **b**?

0.3.2 Answers:

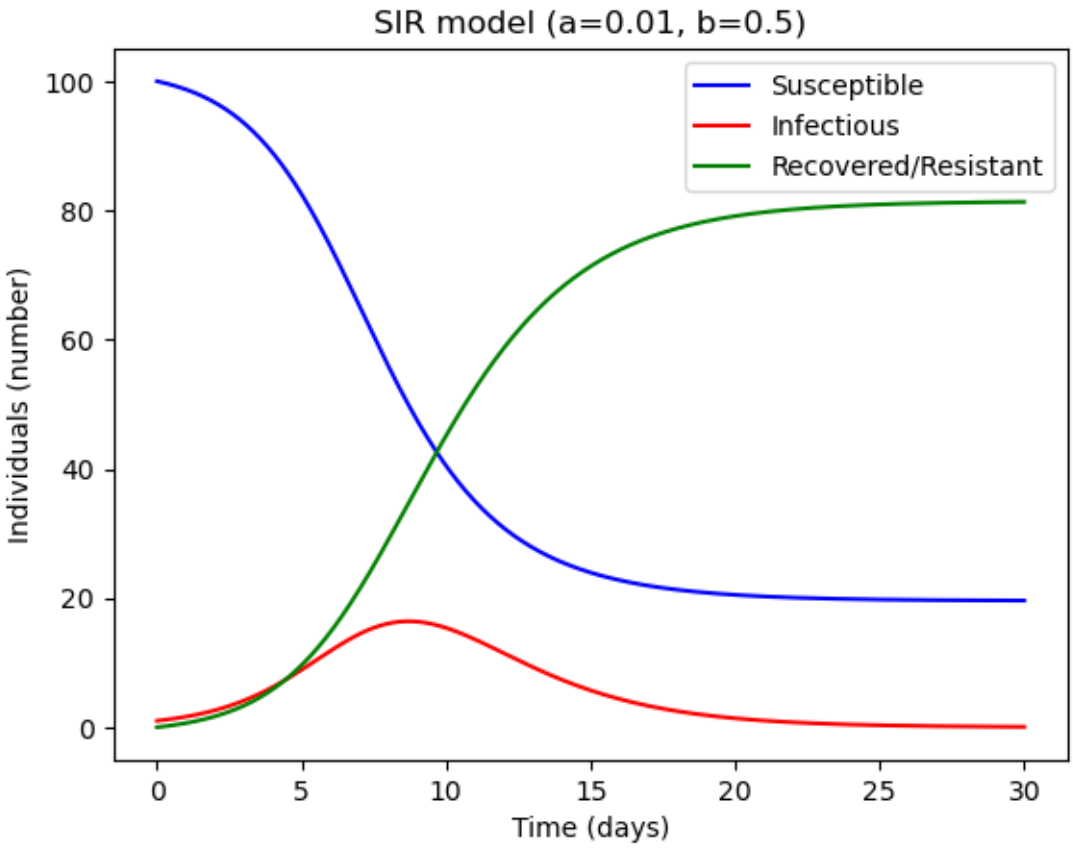
- Increasing a makes I increase faster and reach a higher peak sooner. Decreasing a makes I increase more slowly and lowers the peak, which it reaches later. Eventually the peak happens far enough in the future that I need to change the time scale to see it (the peak is still there, though)
- Increasing b makes the peak of I lower, but doesn't seem to change the timing of the peak. Decreasing b makes the peak higher, again without changing the timing.
- Yes, a low enough a or a high enough b mean some people never get sick. Increasing the threshold b/a above S(0) means the disease never really spreads much.

- Hand washing, masking, social distancing, quarantine, and basic biology of the disease all affect a. Access to care and the biology of the illness are probably the big factors controlling b. Vaccinations may affect both: the COVID vaccines didn't necessarily prevent illness, but they probably made it harder to catch COVID and helped people recover more quickly.

[46]:
```
sir_model(0.0025, 0.125, 100, 1, 0, 100, 1000)
```



[65]:
```
sir_model(0.01, 0.5, 100, 1, 0, 30, 1000)
```


SIR model (a=0.01, b=0.5)
Susceptible
Infectious
Recovered/Resistant
Individuals (number)
Time (days)
0
20
40
60
80
100
0
5
10
15
20
25
30

[ ]: