Group: Amalia Bogri, Christian Berrig & Jonas Bolduan

Covid-19 epidemic by SIR models

a.

Parameters:

Average period of infectivity: $T = 14\ [days]$

Initial number of susceptibles: $S_0 = 6 \cdot 10^6\ [persons]$

Initial number of infected: $I_0 = 15\ [persons]$

Initial number of recovered: $R_0 = 0\ [persons]$

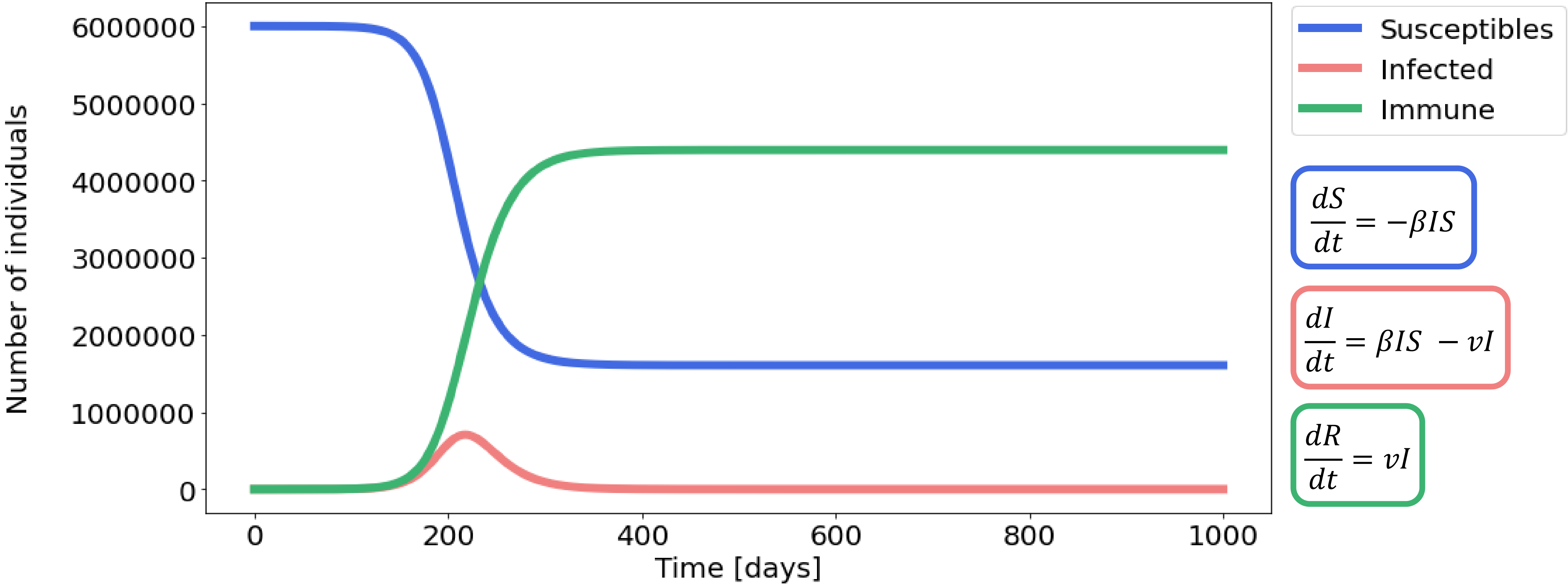
Total population: $N = S_0 + I_0 + R_0$

Intrinsic reproductive number: $Q_0 = 1.8\ [persons]$

Recovery rate: $v = {}^1/_T = {}^1/_{14} \approx 0.07\ [1/days]$

Transmission rate: $\beta = Q_0 \cdot v/\mathrm{N} \approx 2.1 \cdot 10^{-8}\ [1/days]$

Duration of simulation: $t_{max} = 1000$ [days]



$$\frac{dS}{dt} = -\beta IS$$

$$\frac{dI}{dt} = \beta IS - vI$$

$$\frac{dR}{dt} = vI$$

b.

Parameters:

Same as before ($\beta \approx 2.1 \cdot 10^{-8} [1/days]$)

Definition of '***duration of epidemic***':

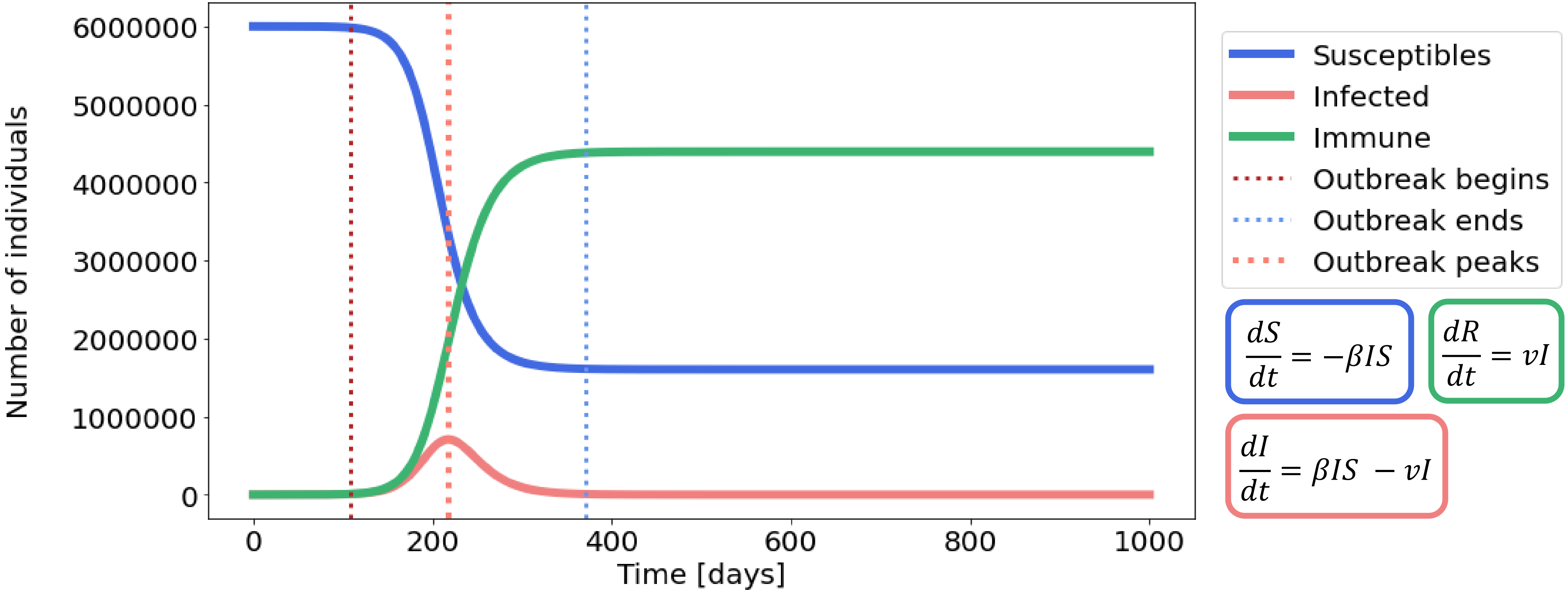
The period when $I(t) > I_0 \cdot 0.01$

Results:

Epidemic lasted: $263\ days$

Epidemic peaked at: day 218 , with $I_{max} = 707359[persons]$

Avoided getting sick: 26.76% of Danish population



$$\frac{dS}{dt} = -\beta IS$$

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

$$\frac{dI}{dt} = \beta IS - \nu I$$

b.

Parameters:

β increased by 10% ($\beta \approx 2.4 \cdot 10^{-8}[1/days]$)

Definition of '*duration of epidemic*':
The period when $I(t) > I_0 \cdot 0.01$

Results:

less!

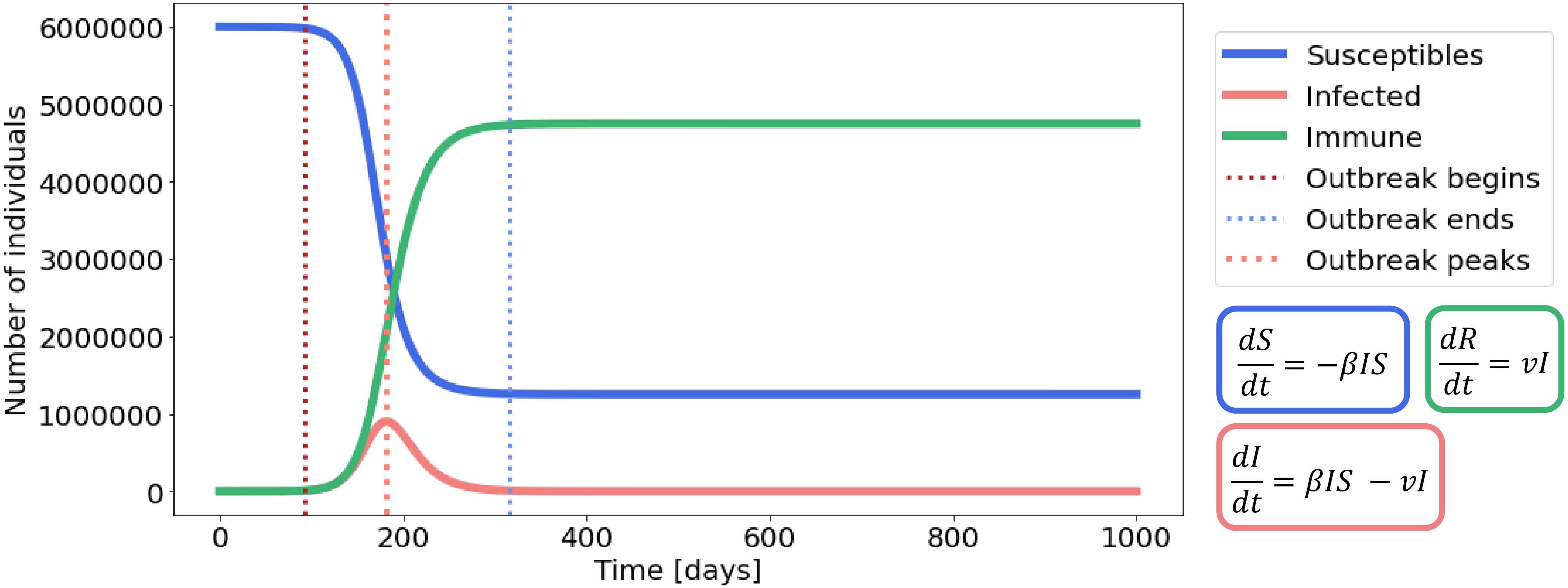
Epidemic lasted: 224 $[days]$

earlier!

Epidemic peaked at: day 182 , with $I_{max} = 899711[persons]$

Avoided getting sick: 20.8% of Danish population

More got sick



$$\frac{dS}{dt} = -\beta IS$$

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

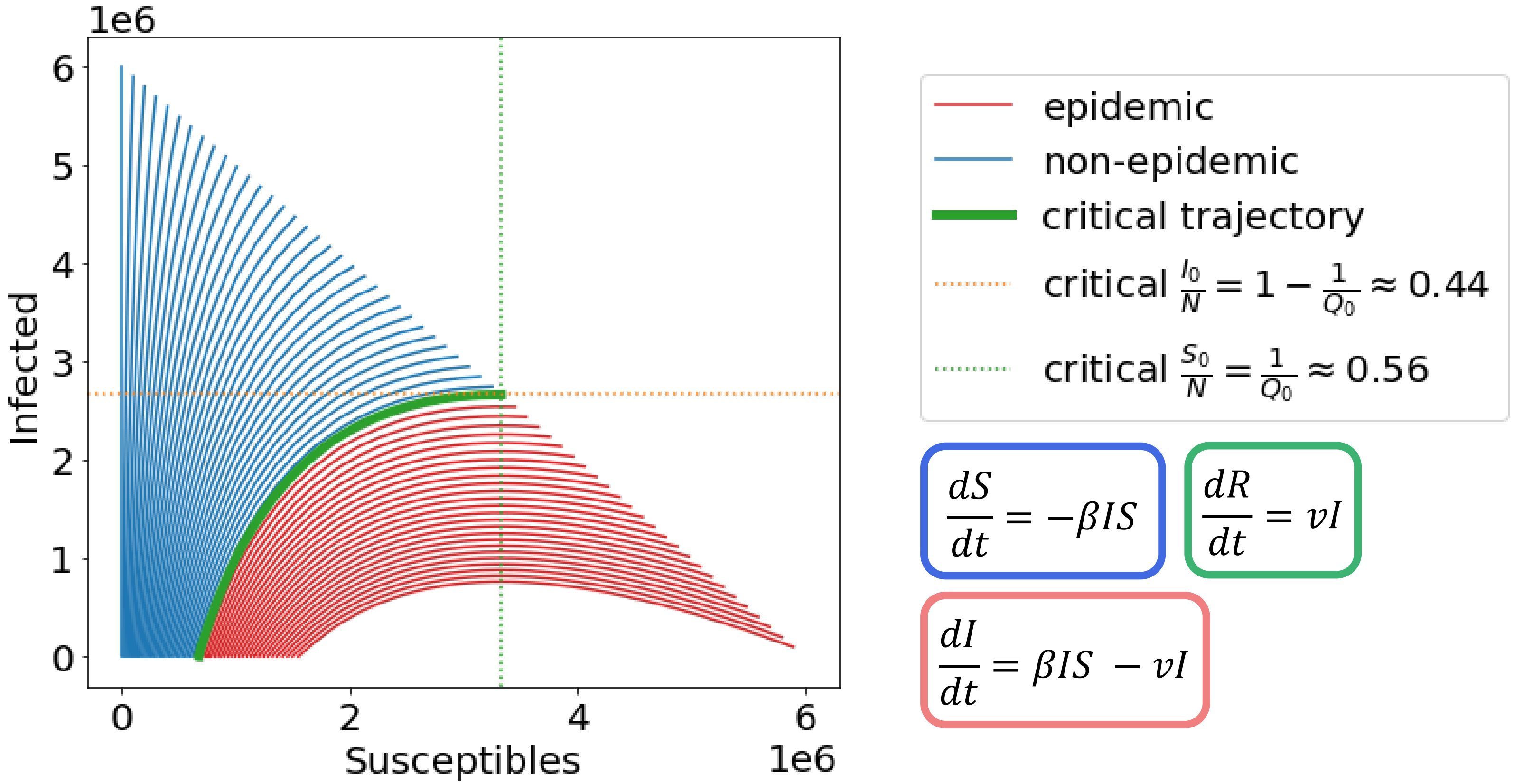
$$\frac{dI}{dt} = \beta IS - \nu I$$

Parameters:

$I_0 \in [0, 6 \cdot 10^6]$

Herd immunity at: $p = 1 - {}^{1}/_{Q0} \approx 0.44$

Critical for immunity $I_0 = \left(1 - {}^{1}/_{Q0}\right) \cdot N = \mathbf{2640000}\ [persons]$



$$\frac{dS}{dt} = -\beta IS$$

$$\frac{dR}{dt} = vI$$

$$\frac{dI}{dt} = \beta IS - vI$$

Parameters:

Same as before ($\beta \approx 2.1 \cdot 10^{-8} [1/days]$)

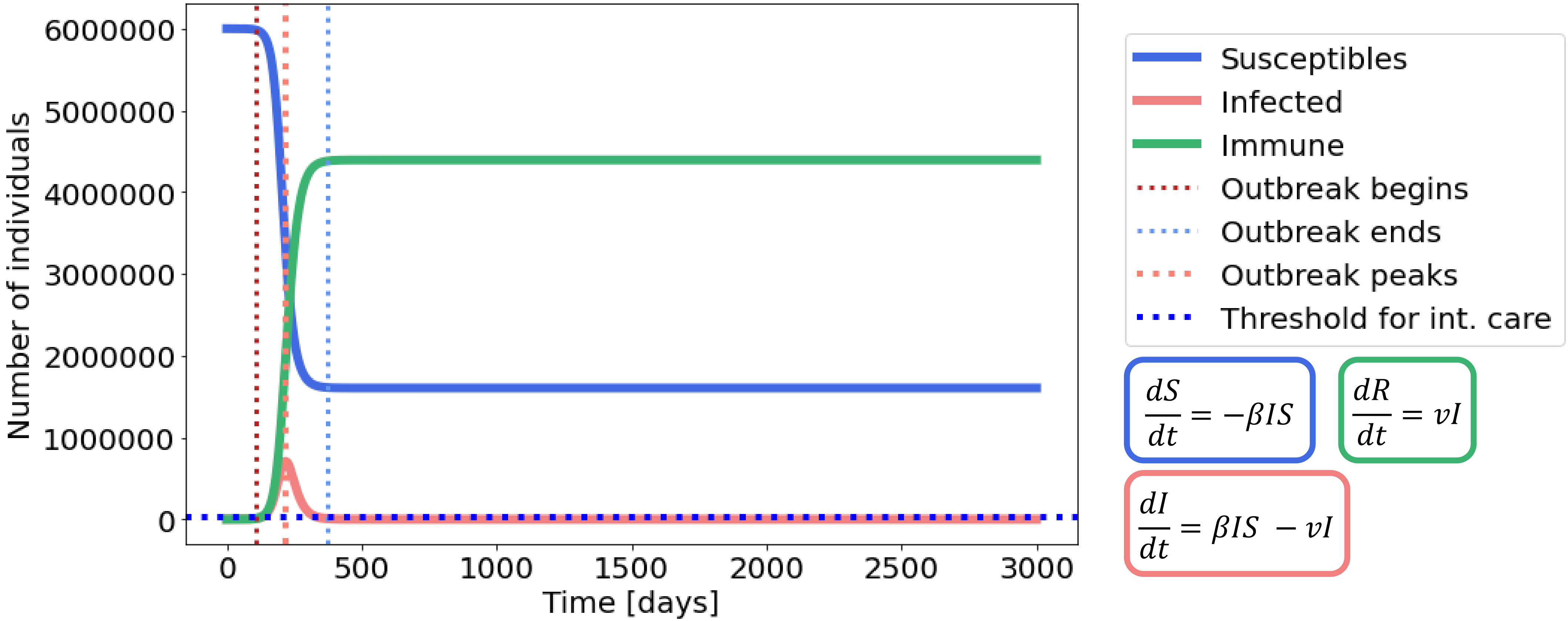
Intensive care patient capacity: $K = 900$ [humans]

Fraction of I that needs int. care: $h = 0.05$ **(i.e. 5% of infected)**

Duration of simulation: $t_{max} = 3000$ [days]

Results:

With the initial conditions, **not all** critically ill patients can get admitted to intensive care at the peak of the epidemic ($I_{cr} = 35368$).



$$\frac{dS}{dt} = -\beta IS$$

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

$$\frac{dI}{dt} = \beta IS - \nu I$$

d.

Parameters:

β reduced to $0.6 \cdot \beta$ (so, $\boldsymbol{\beta \approx 1.2 \cdot 10^{-8}}[1/days]$)

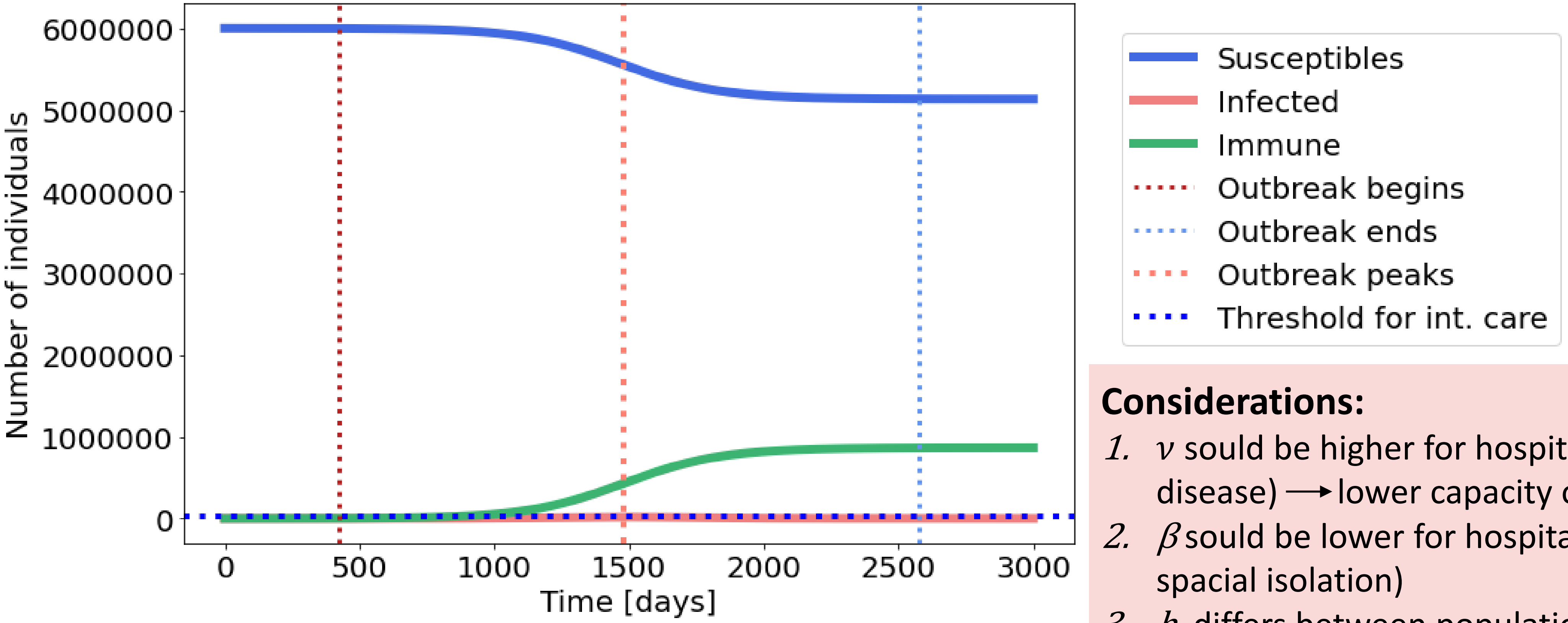
Intensive care patient capacity: $\boldsymbol{K = 900\ [\text{humans}]}$

Fraction of I that needs int. care: $\boldsymbol{h = 0.05}$ **(i.e. 5% of infected)**

Duration of simulation: $t_{max} = 3000$ [days]

Results:

With the new β, **all** critically ill patients can get admitted to intensive care at the peak of the epidemic ($I_{cr} = 844$).



Considerations:

1. ν sould be higher for hospitalized (longer disease) ⟶ lower capacity of intensive care
2. β sould be lower for hospitalized (due to higher spacial isolation)
3. h differs between populations & health systems

e.

Parameters:

Same as initial.

Immunity duration: $T_{im} = 365/2$ (half a year)

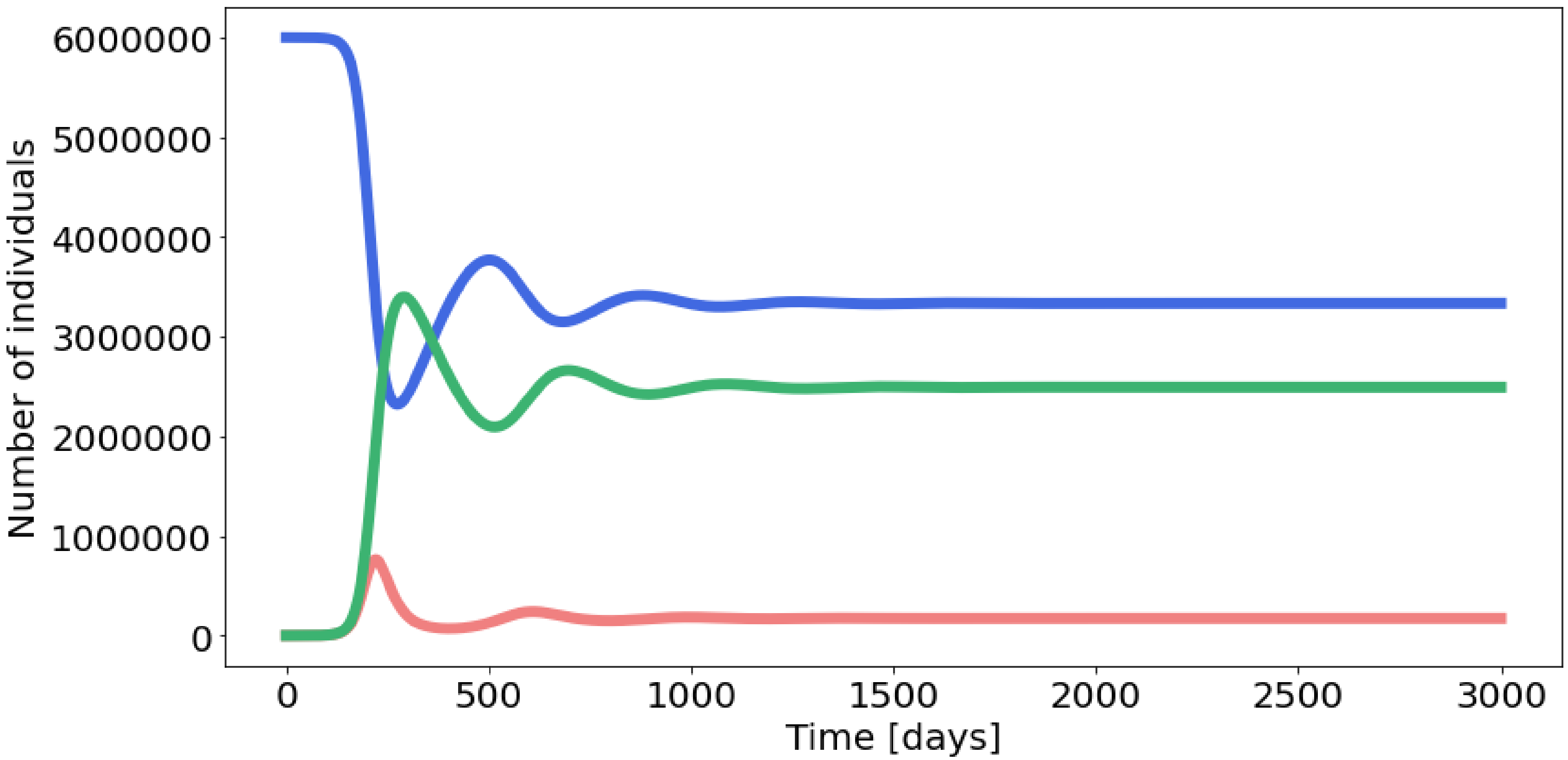
Loss of immunity rate: $\boldsymbol{\gamma = 1/T_{im} = 0.05}$

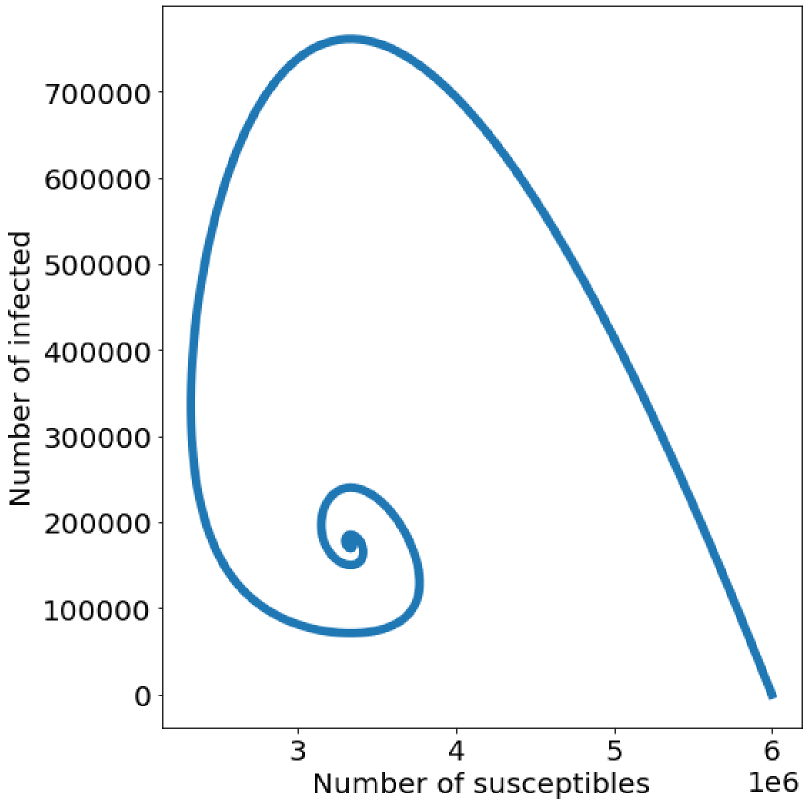
Duration of simulation: $t_{max} = 3000$ [days]

$$\frac{dS}{dt} = -\beta IS + \gamma R$$

$$\frac{dI}{dt} = \beta IS - \nu I$$

$$\frac{dR}{dt} = \nu I - \gamma R$$





Parameters:

Same as before, with loss of immunity.

Seasonality: β **is a function of time** $\varphi(t)$

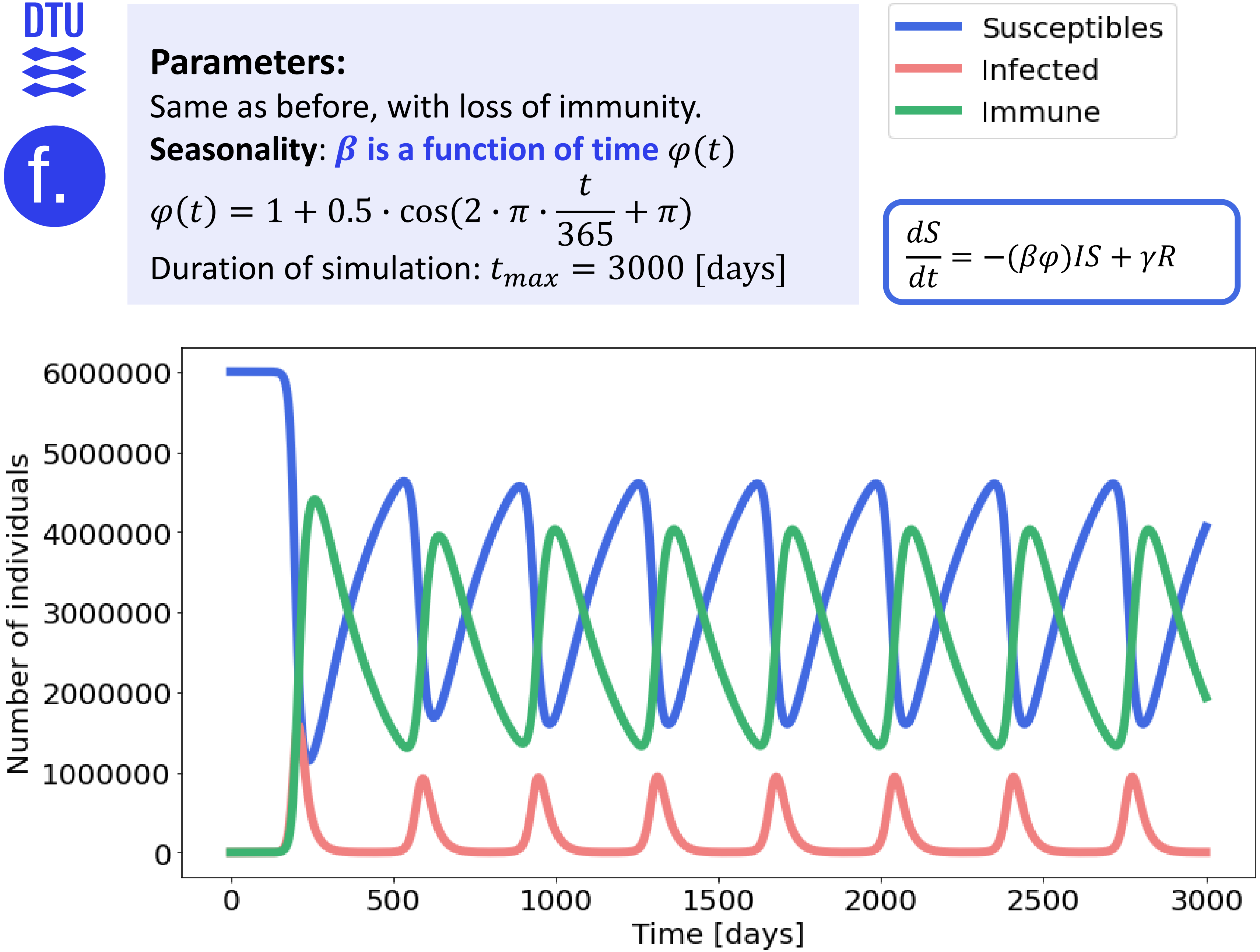
$$\varphi(t) = 1 + 0.5 \cdot \cos(2 \cdot \pi \cdot \frac{t}{365} + \pi)$$

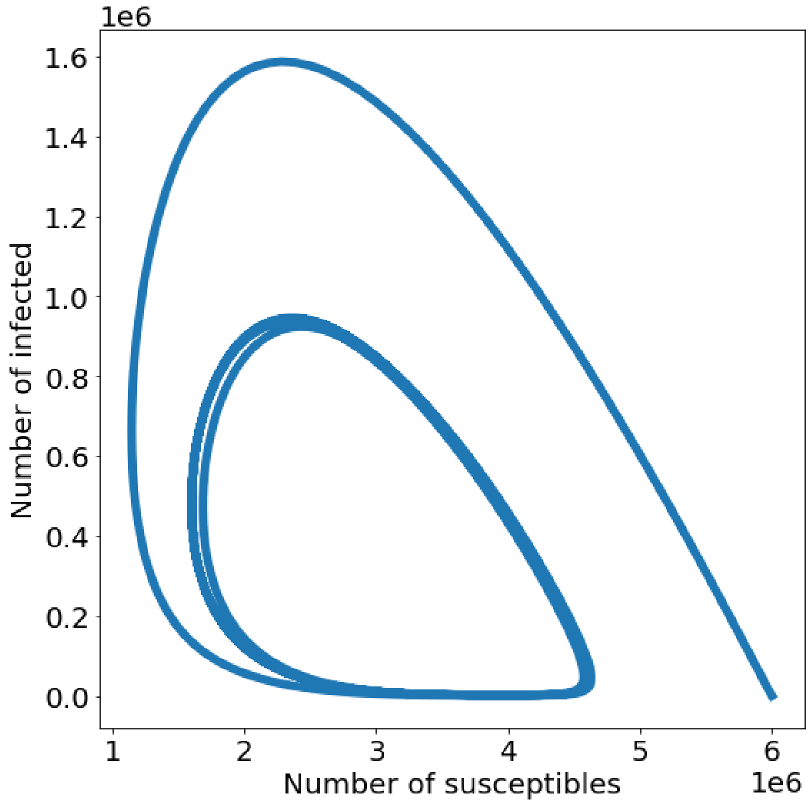
Duration of simulation: $t_{max} = 3000$ [days]

$$\frac{dS}{dt} = -(\beta\varphi)IS + \gamma R$$

$$\frac{dI}{dt} = (\beta\varphi)IS - \nu I$$

$$\frac{dR}{dt} = \nu I - \gamma R$$





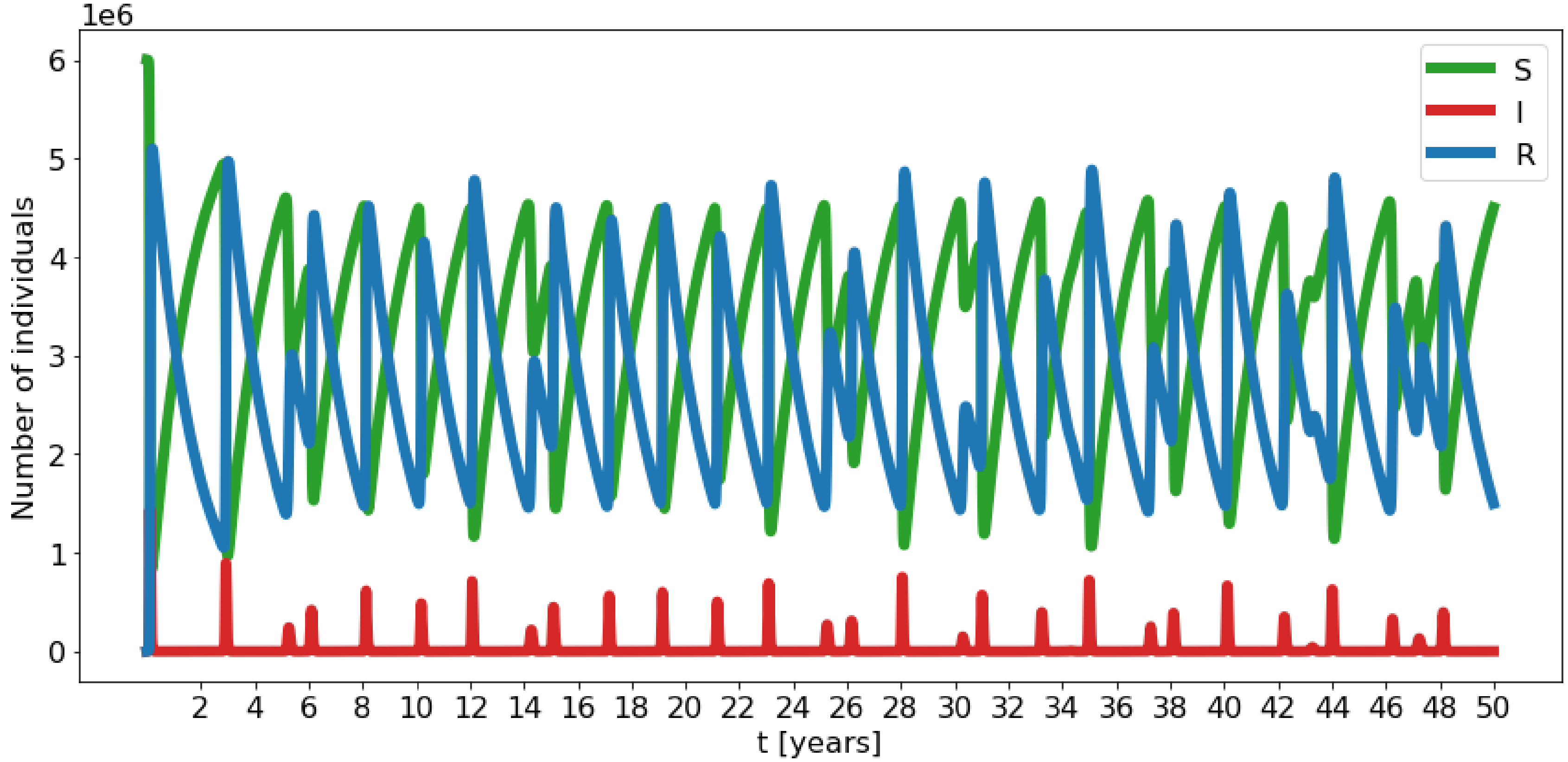
Interesting case of seasonality:
With a (very) different set of initial parameters, we observed chaotic behaviour, and a strange attractor behaviour in the phase plane.

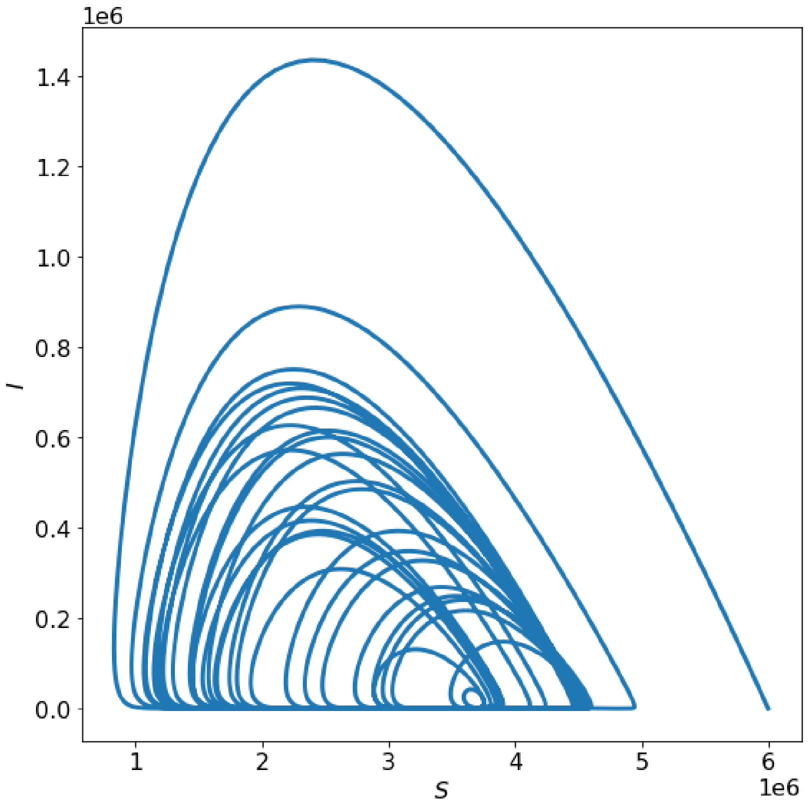
Susceptibles
Infected
Immune

$$\frac{dS}{dt} = -(\beta\varphi)IS + \gamma R$$

$$\frac{dI}{dt} = (\beta\varphi)IS - \nu I$$

$$\frac{dR}{dt} = \nu I - \gamma R$$





DTU

g.

Parameters:

Same as before, with loss of immunity.

Perceived risk: **β is a function of S, $\zeta(S)$**

$\zeta(S) = S/5 \cdot 10^5$

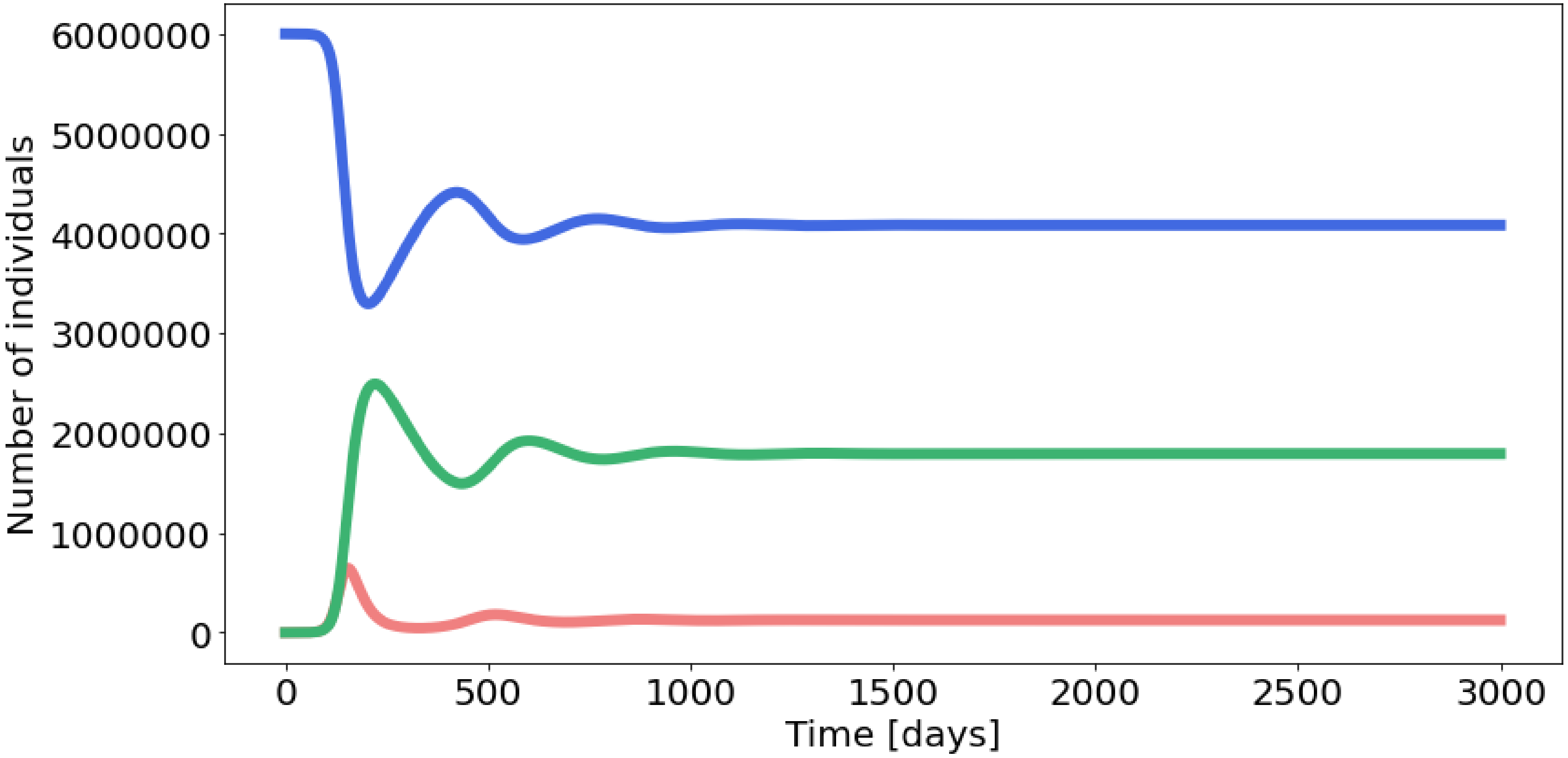
When S decrease (i.e. I increases),
β decreases (i.e. we have less transmission).

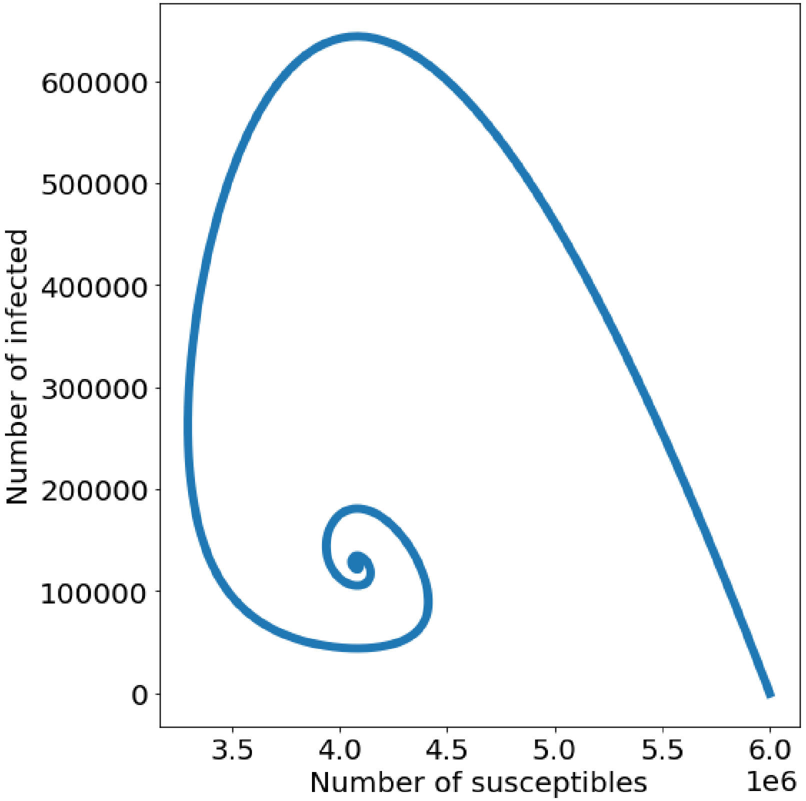
Susceptibles
Infected
Immune

$$\frac{dS}{dt} = -(\beta\zeta)IS + \gamma R$$

$$\frac{dI}{dt} = (\beta\zeta)IS - vI$$

$$\frac{dR}{dt} = vI - \gamma R$$





SIR with vital dynamics:

Same as initial.

We introduce new rate to account for deaths:

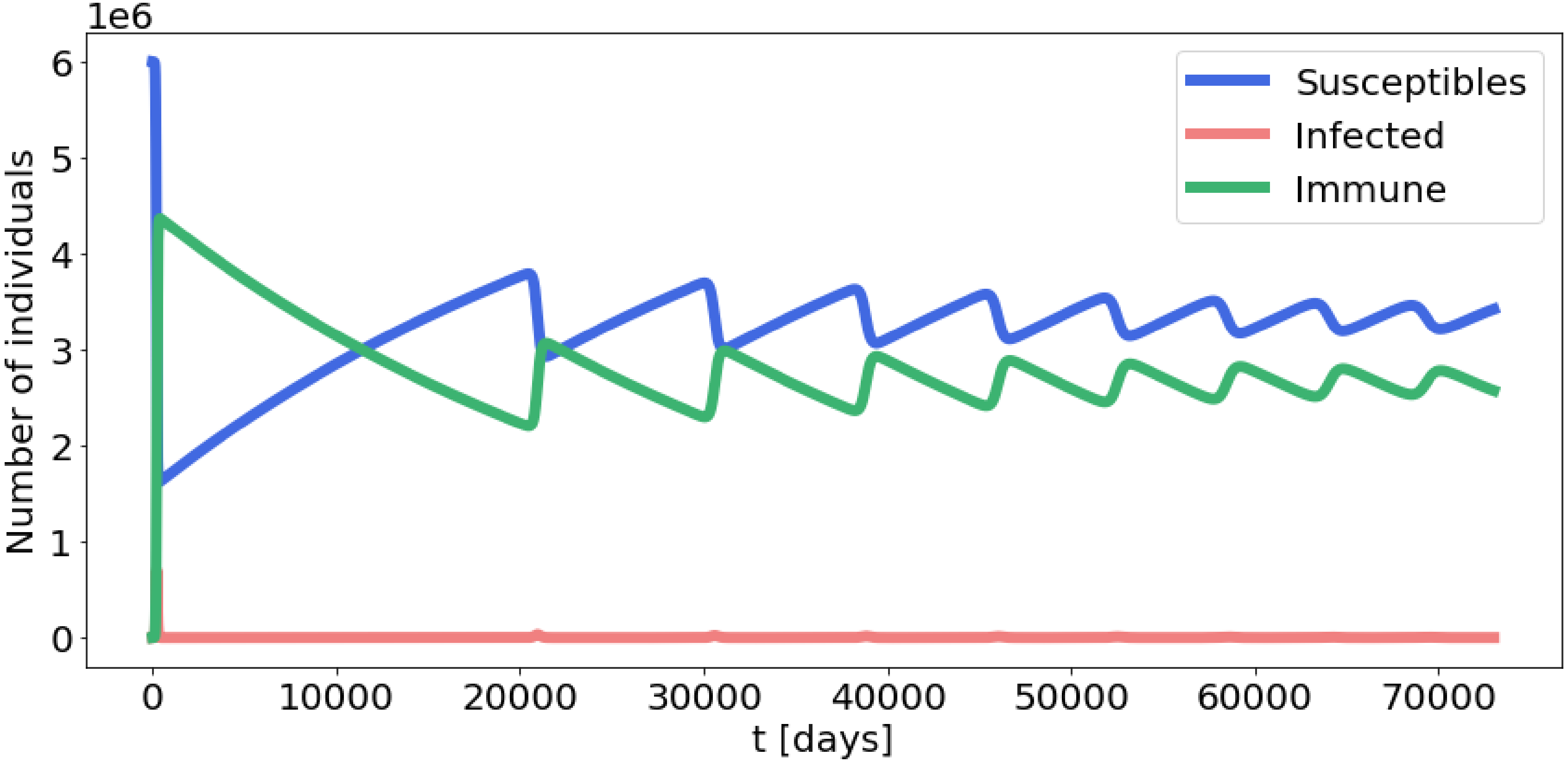
$\mu = 1/(80 \cdot 365)$

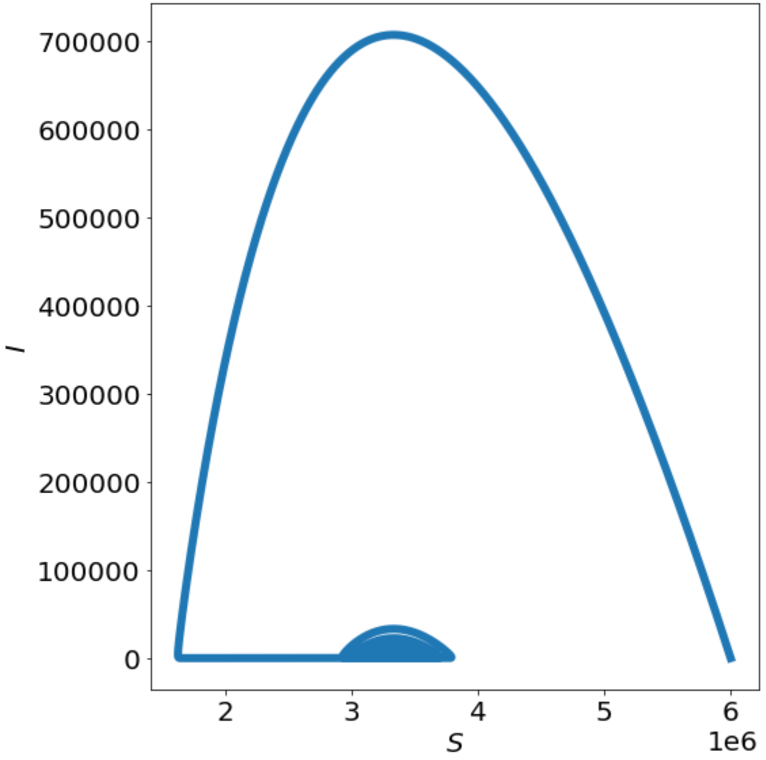
Duration of simulation: $t_{max} = 70000$ [days]

$$\frac{dS}{dt} = \mu N - \beta IS - \mu S$$

$$\frac{dI}{dt} = \beta IS - \nu I - \mu I$$

$$\frac{dR}{dt} = \nu I - \mu R$$





Code in Python:

a.

```
# Import modules:
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import numpy as np

# Define a function with the three SIR equations:
def S_I_R(z, t, b, v):
    S, I, R = z
    dSdt = -b * S * I
    dIdt = b * S * I - v * I
    dRdt = v * I
    dzdt = np.array([dSdt, dIdt, dRdt])
    return dzdt

# Parameters
T = 14 # average period of infectivity
S0 = 6e6 # total susceptibles at the beginning of pandemic in Denmark
I0 = 15 # initial number of infected
R0 = 0 # initial number of recovered
N0 = S0 + I0 + R0 # total population number
z0 = np.array([S0, I0, R0]) # put all the initial conditions in an array
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark

v = 1/T # calculate recovery rate (v)
b = Q0 * v / N0 # calculate transmission rate (b)
par = (b, v) # put parameters in a tuple to use in the ODE

t_max = 1000 # days
t = np.linspace(0, t_max, t_max + 1) # range for time
# I used t_max + 1 for the step number so that I get exactly t_max days
as timesteps, without funny decimal points

# Solve the differential equation
ns = odeint(S_I_R, z0, t, args = par)
S, I, R = ns.T

clr = ['royalblue', 'lightcoral', 'mediumseagreen']
lbl = ['Susceptibles', 'Infected', 'Immune']
```

```
fig, ax = plt.subplots(figsize=(16,6), tight_layout=True)
for i, c in enumerate(ns.T):
    ax.plot(t, c, color = clr[i], label = lbl[i], linewidth=6)
ax.ticklabel_format(useOffset=False, style='plain')
ax.set_xlabel('Time [days]')
ax.set_ylabel('Number of individuals\n')
ax.legend(loc='upper center', bbox_to_anchor=(1.15, 1), borderaxespad=1)
print_parameters()


# Calculate outbreak numbers:
outbreak_start = np.amin(np.where(I > np.amax(I)*0.01))
outbreak_end = np.amax(np.where(I > np.amax(I)*0.01))
outbreak_peak = t[np.argmax(I)] # when does it peak?
Imax = np.int(np.amax(I)) # maximum infected
outbreak_duration = outbreak_end  - outbreak_start # how long does it last?
not_sick = np.int(np.amin(S)) # number of susceptible people at equilibrium
not_sick_percentage = np.round(100 * not_sick / N0, 2) # find the persentage of not sick and round to 2
decimals
equilibrium_start = np.amin(np.where(S < np.int((np.amin(S))+1))) # day when equations reach equilibrium

def print_parameters():
    print(f'Transmission rate β: {b}.')
    print(f'Epidemic peaks at day {np.int(outbreak_peak)}.')
    print(f'At peak, {Imax} persons were infected per day.')
    print(f'Epidemic lasts {outbreak_duration} days (I > 0.01 of Imax).')
    print(f'{not_sick} persons did not get sick, i.e. {not_sick_percentage}% of the Danish population.')
```

b.

```
fig, ax = plt.subplots(figsize=(16,6), tight_layout=True)
for i, c in enumerate(ns.T):
    ax.plot(t, c, color = clr[i], label = lbl[i], linewidth=6)
ax.axvline(outbreak_start, color='firebrick', linestyle=':',linewidth=3, label='Outbreak begins')
ax.axvline(outbreak_end, color='cornflowerblue', linestyle=':',linewidth=3, label='Outbreak ends')
ax.axvline(outbreak_peak, color='salmon', linestyle=':',linewidth=4, label='Outbreak peaks')
ax.ticklabel_format(useOffset=False, style='plain') # just to show the full numbers and not raised to power
ax.set_xlabel('Time [days]')
ax.set_ylabel('Number of individuals\n')
ax.legend(loc='upper center', bbox_to_anchor=(1.2, 1), borderaxespad=1)
print_parameters()

# For the 10% increase we did exactly the same as above just using b_incr instead of b:
b_incr = b + b*0.1
par = (b_incr, v)
# I will not copy paste the same code
```

Code in Python: c.

```
# c. Phase space plot:
# Parameters
N = 6e6 # total population number
T = 14 # average period of infectivity
R0 = 0 # initial number of recovered
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark
v = 1/T # calculate recovery rate (v)
b = Q0 * v / N0 # calculate transmission rate
t_max = 1000 # days
t = np.linspace(0, t_max, t_max + 1) # range for time

print(f"heard immunity at I=1-1/Q0={1-1/Q0}")

fig, ax = plt.subplots(figsize=(14,6), tight_layout=True)
I0s = np.linspace(0, int(6e6), 60)
for I0 in I0s:
    S0 = N-I0 # total susceptibles at the beginning of pandemic in Denmark
    col_cond = (I0/N >= 1-1/Q0)
    c = int(col_cond)*"tab:blue" + (1-int(col_cond))*"tab:red"

    #state_init = np.array([N-I0, I0, 0])
    z0 = np.array([S0, I0, R0])
    params = (b, v)

    num_sol = odeint(S_I_R, z0, t, args=params).T
    S, I, R = num_sol
    final_size = 1 - S[-1]/N

    lab = int(I0 == I0s[0])*"epidemic"+int(I0 == I0s[-1])*"non-epidemic"
    ax.plot(S, I, color=c, label=lab)
```

```
heard_immune = (1-1/Q0)
I0 = (1-1/Q0)*N
S0 = N-I0 # total susceptibles at the beginning of pandemic in Denmark
z0 = np.array([S0, I0, R0])
params = (b, v)

num_sol = odeint(S_I_R, z0, t, args=params).T
S, I, R = num_sol
ax.plot(S, I, color="tab:green", linewidth=5, alpha=1,
        label="critical trajectory")
ax.axhline(N*(1-1/Q0), color="tab:orange", linestyle=":",
           label="critical $\\frac{I_{0}}{N}=1-\\frac{1}{Q_{0}} \\approx $" +
str(round(1-1/Q0,2)))
ax.axvline(N/Q0, color="tab:green", linestyle=":",
           label="critical $\\frac{S_{0}}{N}=\\frac{1}{Q_{0}}
\\approx$"+str(round(1/Q0,2)))

ax.set_xlabel("Susceptibles")
ax.set_ylabel("Infected")
ax.legend(loc='upper center', bbox_to_anchor=(1.5, 1), borderaxespad=1)
```

Code in Python: d.

```python
# d. Intensive care

# Parameters
T = 14 # average period of infectivity
S0 = 6e6 # total susceptibles at the beginning of pandemic in Denmark
I0 = 15 # initial number of infected
R0 = 0 # initial number of recovered
N0 = S0 + I0 + R0 # total population number
z0 = np.array([S0, I0, R0]) # put all the initial conditions in an array
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark
v = 1/T # calculate recovery rate (v)

b = (Q0 * v / S0) # I just tried different numbers! not sure if this is what they
wanted!
par = (b, v) # put parameters in a tuple to use in the ODE

t_max = 3000
t = np.linspace(0, t_max, t_max + 1) # range for time

K = 900 # intensive care patient capacity
h = 0.05 # 5% of infected need intensive care

# Solve the differential equation
ns = odeint(S_I_R, z0, t, args = par)
S, I, R = ns.T

# Calculate outbreak numbers:
outbreak_start = np.amin(np.where(I > np.amax(I)*0.01))
outbreak_end = np.amax(np.where(I > np.amax(I)*0.01))
outbreak_peak = t[np.argmax(I)]
Imax = np.int(np.amax(I))
outbreak_duration = outbreak_end  - outbreak_start
not_sick = np.int(np.amin(S))
not_sick_percentage = np.round(100 * not_sick / N0, 2)
equilibrium_start = np.amin(np.where(S < np.int((np.amin(S))+1)))
```

```python
# Plot:
fig, ax = plt.subplots(figsize=(16,6), tight_layout=True)
for i, c in enumerate(ns.T):
    ax.plot(t, c, color = clr[i], label = lbl[i], linewidth=6)

ax.axvline(outbreak_start, color='firebrick', linestyle=':',linewidth=3, label='Outbreak
begins')
ax.axvline(outbreak_end, color='cornflowerblue', linestyle=':',linewidth=3,
label='Outbreak ends')
ax.axvline(outbreak_peak, color='salmon', linestyle=':',linewidth=4, label='Outbreak
peaks')
ax.axhline(18000, color='blue', linestyle=':',linewidth=4, label='Threshold for int.
care')


ax.ticklabel_format(useOffset=False, style='plain')
ax.set_xlabel('Time [days]')
ax.set_ylabel('Number of individuals')
ax.legend(loc='upper center', bbox_to_anchor=(1.3, 1), borderaxespad=1) # place legend
outside the graph

print_parameters()

critically_ill = np.int(Imax * h)

if critically_ill <= K:
    print(f'All critically ill patients ({critically_ill}) can get intentsive care')
else:
    print(f'Not all critically ill patients ({critically_ill}) can get intensive care')
```

```python
# Then, we did the same for b = (Q0 * v / S0)*0.6, and I am not showing this code
```

Code in Python: e.

```
# e. Loss of Immunity

# Define a function with the three SIR equations:
def S_I_R_immunity(z, t, b, v, g):
    S, I, R = z
    dSdt = -b * S * I + g * R
    dIdt = b * S * I - v * I
    dRdt = v * I - g * R
    dzdt = np.array([dSdt, dIdt, dRdt])
    return dzdt


# Parameters
T = 14 # average period of infectivity
S0 = 6e6 # total susceptibles at the beginning of pandemic in Denmark
I0 = 15 # initial number of infected
R0 = 0 # initial number of recovered
N0 = S0 + I0 + R0 # total population number
z0 = np.array([S0, I0, R0]) # put all the initial conditions in an array
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark
v = 1/T # calculate recovery rate (v)
b = Q0 * v / N0 # calculate transmission rate (b) with data from the beginning of
the pandemic
g = 0.005 # 1/(365/2)
par = (b, v, g) # put parameters in a tuple to use in the ODE

t_max = 3000 # days
t = np.linspace(0, t_max, t_max + 1) # range for time


# Solve the differential equation
ns = odeint(S_I_R_immunity, z0, t, args = par)
S, I, R = ns.T
```

```
# Calculate outbreak numbers:
outbreak_start = np.amin(np.where(I > np.amax(I)*0.01))
outbreak_end = np.amax(np.where(I > np.amax(I)*0.01))
outbreak_peak = t[np.argmax(I)]
Imax = np.int(np.amax(I))
outbreak_duration = outbreak_end  - outbreak_start
not_sick = np.int(np.amin(S))
not_sick_percentage = np.round(100 * not_sick / N0, 2)
equilibrium_start = np.amin(np.where(S < np.int((np.amin(S))+1)))

# Plot:
fig, ax = plt.subplots(figsize=(16,6), tight_layout=True)
for i, c in enumerate(ns.T):
    ax.plot(t, c, color = clr[i], label = lbl[i], linewidth=6)

ax.ticklabel_format(useOffset=False, style='plain')
ax.set_xlabel('Time [days]')
ax.set_ylabel('Number of individuals')
ax.legend(loc='upper center', bbox_to_anchor=(1.2, 1), borderaxespad=1) # place legend
outside the graph
plt.show()

print_parameters()

critically_ill = np.int(Imax * h)

if critically_ill <= K:
    print(f'All critically ill patients ({critically_ill}) can get intentsive care')
else:
    print(f'Not all critically ill patients ({critically_ill}) can get intensive care')

# Making phase-plane plot:
fig, ax = plt.subplots(figsize=(8,8), tight_layout=True)
ax.ticklabel_format(useOffset=True)
ax.plot(S, I, linewidth = 6)
ax.set_xlabel("Number of susceptibles")
ax.set_ylabel("Number of infected")
plt.show()
```

Code in Python: f.

```
# f. Seasonality

# Define a function with the three SIR equations:
def S_I_R_seasonality(z, t, b, v, g):
    #season = 0.31*(np.sin(((t/365)*2*np.pi)+200)+2) # amalia's
    season = 1 + 0.5*np.cos(2*np.pi*t*1/365 + np.pi) # christian's
    S, I, R = z
    dSdt = -(b*season) * S * I + g * R
    dIdt = (b*season) * S * I - v * I
    dRdt = v * I - g * R
    dzdt = np.array([dSdt, dIdt, dRdt])
    return dzdt



# Parameters
T = 14 # average period of infectivity
S0 = 6e6 # total susceptibles at the beginning of pandemic in Denmark
I0 = 15 # initial number of infected
R0 = 0 # initial number of recovered
N0 = S0 + I0 + R0 # total population number
z0 = np.array([S0, I0, R0]) # put all the initial conditions in an array
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark
v = 1/T # calculate recovery rate (v)
b = Q0 * v / N0 # calculate transmission rate (b) with data from the beginning of
the pandemic
g = 0.005 # 1/(365/2)
par = (b, v, g) # put parameters in a tuple to use in the ODE

t_max = 3000 # days
t = np.linspace(0, t_max, t_max + 1) # range for time

# Solve the differential equation
ns = odeint(S_I_R_seasonality, z0, t, args = par)
S, I, R = ns.T
```

```
# Calculate outbreak numbers:
outbreak_start = np.amin(np.where(I > np.amax(I)*0.01))
outbreak_end = np.amax(np.where(I > np.amax(I)*0.01))
outbreak_peak = t[np.argmax(I)]
Imax = np.int(np.amax(I))
outbreak_duration = outbreak_end  - outbreak_start
not_sick = np.int(np.amin(S))
not_sick_percentage = np.round(100 * not_sick / N0, 2)
equilibrium_start = np.amin(np.where(S < np.int((np.amin(S))+1)))


# Plot:
fig, ax = plt.subplots(figsize=(16,6), tight_layout=True)
for i, c in enumerate(ns.T):
    ax.plot(t, c, color = clr[i], label = lbl[i], linewidth=6)

ax.ticklabel_format(useOffset=False, style='plain')
ax.set_xlabel('Time [days]')
ax.set_ylabel('Number of individuals')
ax.legend(loc='upper center', bbox_to_anchor=(1.2, 1), borderaxespad=1)
plt.show()

print_parameters()

critically_ill = np.int(Imax * h)

if critically_ill <= K:
    print(f'All critically ill patients ({critically_ill}) can get intentsive care')
else:
    print(f'Not all critically ill patients ({critically_ill}) can get intensive care')

# Making phase-plane plot:
fig, ax = plt.subplots(figsize=(8,8), tight_layout=True)
ax.plot(S, I, linewidth = 6)
ax.set_xlabel("Number of susceptibles")
ax.set_ylabel("Number of infected")
plt.show()
```

Code in Python: j.

```python
# (j) SIR w. vital dynamics
T = 14 # average period of infectivity
S0 = 6e6 # total susceptibles at the beginning of pandemic in Denmark
I0 = 15 # initial number of infected
R0 = 0 # initial number of recovered
N0 = S0 + I0 + R0 # total population number
z0 = np.array([S0, I0, R0]) # put all the initial conditions in an array
Q0 = 1.8 # reproductive number at the start of the pandemic in Denmark
v = 1/T # calculate recovery rate (v)
b = Q0 * v / N0 # calculate transmission rate (b) with data from the beginning of
the pandemic
g = 0.005 # 1/(365/2)
par = (b, v, g) # put parameters in a tuple to use in the ODE

yr = 365
mu = 1/(80*yr)

I0 = 15

t_max = yr*200
t = np.linspace(0, t_max, 50*t_max) # range for time

def deriv(state, t, beta, nu, mu):
    S, I, R = state
    dS_dt = mu*N - b*S*I - mu*S
    dI_dt = b*S*I - v*I - mu*I
    dR_dt = v*I - mu*R
    return np.array([dS_dt, dI_dt, dR_dt])

z0 = np.array([S0, I0, R0])
params = (b, v, mu)

num_sol = odeint(deriv, z0, t, args=params).T
S, I, R = num_sol
final_size = 1 - S[-1]/N
```

```python
# Making a nice plot:
fig, ax = plt.subplots(figsize=(12,6), tight_layout=True)
for i, c in enumerate(num_sol):
    ax.plot(t, c, color=clr[i], label=lbl[i], linewidth=6)

ax.set_xlabel("t [days]")
ax.set_ylabel("Number of individuals")
# ax.set_yscale("log")
ax.legend()

# Making phase-plane plot:
fig, ax = plt.subplots(figsize=(8,8), tight_layout=True)
ax.plot(S, I, linewidth=6)

ax.set_xlabel("$S$")
ax.set_ylabel("$I$")
# ax.set_yscale("log")
# ax.legend()
```