# FFR110 - Homework Problem 3 - Part 1

Computational Biology

**Felix Waldschock**
000420-T398

Chalmers University of Technology
February 2024

# Contents

# 1 Problem 1, Stochastic dynamics in large but finite populations

The SIS-model

$$\frac{dI}{dt} = \frac{\alpha}{S+I} * S * I - \beta * I$$
$$\frac{dS}{dt} = -\frac{\alpha}{S+I} * S * I + \beta * I$$

with $\alpha$ and $\beta$ positive constants. Also $S$ and $I$ only take values $>= 0$, summed $S+I = N$ give the entire population which in this problem is constant.

## 1.1 a) Deterministic model

## Problem 1a

$$S = N - I$$

$$\frac{dI}{dt} = \frac{\alpha}{S+I} S \cdot I - \beta I$$

$$\frac{dI}{dt} = \frac{\alpha}{N} S I - \beta I$$

$$\frac{dI}{dt} = I \cdot \left( \frac{\alpha}{N} S - \beta \right) \qquad \| \; S = N - I \qquad (1)$$

$$\frac{dI}{dt} = I \cdot \left( \frac{\alpha}{N} (N - I) - \beta \right)$$

$$\frac{dI}{dt} \overset{!}{=} 0 \qquad \text{(steady state)}$$

$$0 = I \left( \frac{\alpha}{N} (N - I) - \beta \right)$$

$$I_1^* = 0$$

$$\frac{\alpha}{N} (N - I) = \beta$$

$$I_2^* = N \left( 1 - \frac{\beta}{\alpha} \right)$$

Steady states at $I^* = \left( 0, \; N \left( 1 - \frac{\beta}{\alpha} \right) \right)$

## Stability analysis

expand around steady states:

$$f(I^* + \eta) = f(I^*) + n f'(I^*)$$

with $f(I) = (1) = -\frac{\alpha I^2}{N} + \alpha I - \beta I$

now around $I_1^*$:

$$\dot{n} = n \underbrace{(\alpha - \beta)}_{\tau}$$

if $\tau < 0$ Stable $\Rightarrow \alpha < \beta$

if $\tau > 0$ unstable $\Rightarrow \alpha > \beta$

around $I_2^*$:

$$\dot{\eta} = -n(\alpha - \beta)$$

due to the sign change, conditions flip

stable $\Rightarrow \alpha > \beta$

unstable $\Rightarrow \alpha < \beta$.

## 1.2 b) Stochastic model for finite population size

$n \to n+1$ $\quad$ $b_n = \alpha \cdot n \left(1 - \frac{n}{N}\right) \leftarrow$ infection

(2)

$n \to n-1$ $\quad$ $d_n = \beta n \leftarrow$ recovery

from Lecture notes:

$$\frac{d}{dt} \rho_n(t) = \lambda_{n-1} \rho_{n-1} + \mu_{n+1} \cdot \rho_{n+1} - (\lambda_n + \mu_n) \rho_n$$

describes probability to have n individuals at time t.

new lets write this with (2):

$$\frac{\partial}{\partial t} p_n = b_{n-1} p_{n-1} + d_{n+1} p_{n+1} - (b_n + d_n) p_n \qquad (3)$$

New introduce the new operator as in the Lecture.

$$E^+_{g_n} = g_{n+1} \quad \text{and} \quad E^-_{g_n} = g_{n-1}$$

and also $I' = \frac{n}{N}$. $\qquad$ (4)

Rewrite (3) with the new operator:

$$\frac{\partial}{\partial t} p_n = (E^- - 1) b_n p_n + (E^+ + 1) d_n p_n \qquad (5)$$

with (4) we find:

$p(t) = \mathcal{N} p(I', t)$ $\qquad$ (assume that $p_n$ smooth)

$$b_n = N\,b(I')$$
$$d_n = N\,d(I')$$

$$b(I') = \alpha I'(1-I')$$
$$d(I') = \beta I'$$

this leads to
$$E^{\pm} g(I') = \sum_{k=0}^{\infty} \frac{(\pm\frac{1}{N})^k}{k!} \frac{\partial^k g}{\partial I'^k} = e^{\pm\frac{1}{N}\frac{\partial}{\partial I'}} g(I')$$

$$e^{\pm\frac{1}{N}\frac{\partial}{\partial I'}} \overset{N\to\infty}{\cong} 1 + \frac{1}{N}\frac{\partial}{\partial I'} \qquad (6)$$

put into (5):
$$\frac{\partial}{\partial t} p_n(I',t) = \left[ (e^{-\frac{1}{N}\frac{\partial}{\partial I'}} - 1) N b(I') + (e^{\frac{1}{N}\frac{\partial}{\partial I'}} - 1) N d(I') \right] \rho(I',t)$$

inserting the limit (6):

$$\frac{\partial}{\partial t} \rho(I',t) = -\frac{\partial}{\partial I'} (b(I')\rho(I',t)) + \frac{\partial}{\partial I'} (d(I')\rho(I',t))$$

Now define $V(I') = b(I') - d(I')$

giving:

$$\frac{dI}{dt} = V(I') = b(I') - d(I') = \alpha I'(1-I') - \beta I'$$

In the stochastical model the steady state
$I_1^* = 0$ is an attracting FP. It can be seen as a quasi steady state.

## 1.3 c) Towards efficient simulation of the stochastic model

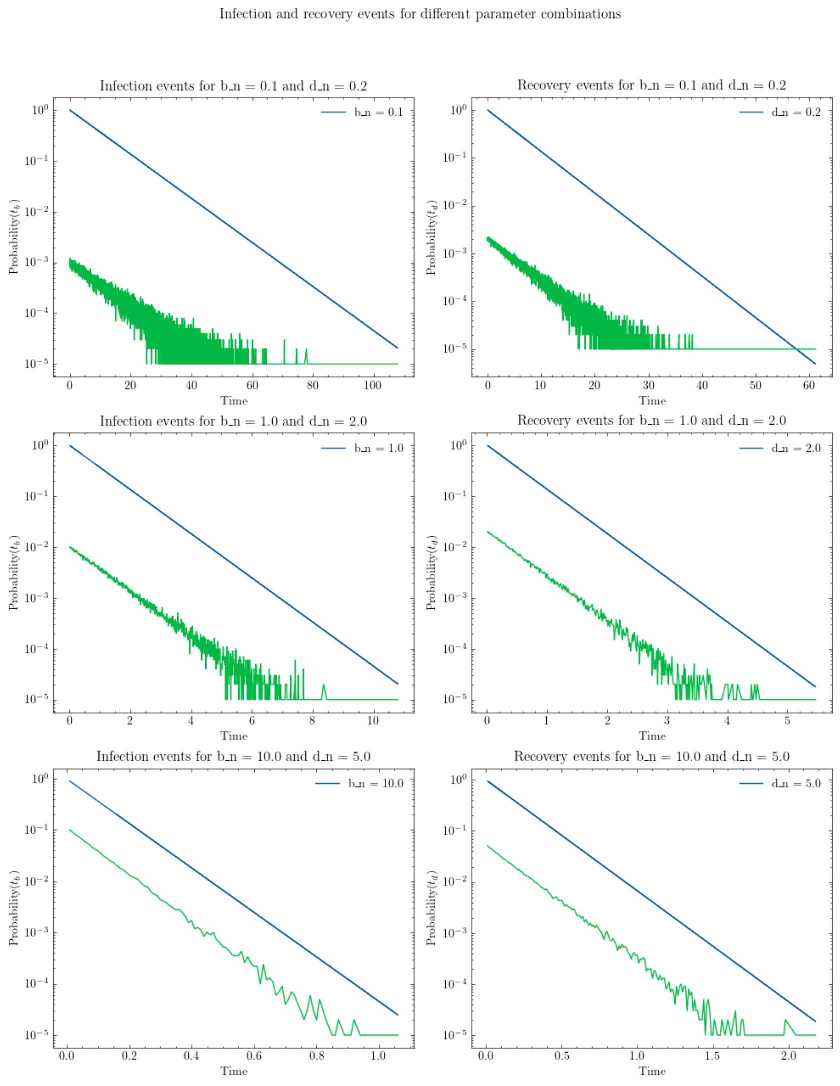

Figure 1: Typical recovery and infections times for the given parameter combinations in the exercise.

The blue lines in the plots are the step (iv) from the exercise to verify that the distributions decay exponentially with $-\lambda * t$. One finds in deed that, both the recover and the infections, do follow an exponential behaviour described by $e^{b_n * t}$ and analogue $e^{d_n * t}$ (trendline of green is parallel to blue). This leads to the following characteristic times it takes for a infection and recovery as display in Table 1.

| **Parameters** | **Infection time [s]** | **Recover time [s]** |
|---|---|---|
| $b_n = 0.1$,$d_n = 0.2$ | 10 | 4.98 |
| $b_n = 1$,$d_n = 2$ | 1 | 0.5 |
| $b_n = 10$,$d_n = 5$ | 0.1 | 0.2 |

Table 1: Average times for a infection / recovery in the SIS model

## List of Figures

## List of Tables

# Untitled-1

```
# %% [markdown]
# # CompBio Problem set 3
#
# ## Problem 1
#

# %%
import numpy as np
import matplotlib.pyplot as plt
import scienceplots
plt.style.use('science')
from tqdm import trange

# %% [markdown]
# SIS - MODEL
#
# $\frac{dI}{dt} = \frac{\alpha}{S+I} * S * I - \beta * I$
# $\frac{dS}{dt} = - \frac{\alpha}{S+I} * S * I + \beta * I$
#

# %%
# Parameters

## infection
bn_values = [0.1,1,10]

## recovery
dn_values = [0.2,2,5]

## parameter combinations
parameters = [[bn_values[0], dn_values[0]], [bn_values[1], dn_values[1]],
[bn_values[2], dn_values[2]]]

# Simumation parameters
dt = 0.01
tEnd = 1000
steps = int(tEnd/dt)
timeVector = np.linspace(0, tEnd, steps)

# %%
def timeOfInterest(value):
    timeOfInterest = dt
    while True:
        if np.random.random() < value:
            return timeOfInterest
        timeOfInterest += dt

# %%
# store for each parameter combination the time t_b and t_d
t_b_values = []
t_d_values = []
```

```python

for i in trange(3):
    b_param = parameters[i][0]
    d_param = parameters[i][1]
    t_b = []
    t_d = []
    for j in range(steps):
        prob_TB = b_param*dt
        prob_TD = d_param*dt
        # n increase
        timeOfInterest_TB = timeOfInterest(prob_TB)
        # n decrease
        timeOfInterest_TD = timeOfInterest(prob_TD)
        t_b.append(timeOfInterest_TB)
        t_d.append(timeOfInterest_TD)

    t_b_values.append(t_b)
    t_d_values.append(t_d)

# %%
# sort the arrays and normalize them to have the probability
t_b_values = np.array(t_b_values)
t_d_values = np.array(t_d_values)

# convert to numpy array
t_b_values = np.array(t_b_values)
t_d_values = np.array(t_d_values)
bn_values = np.array(bn_values)
dn_values = np.array(dn_values)

# compute the Infection and recovery events
bn_values_expanded = bn_values[:, np.newaxis]
dn_values_expanded = dn_values[:, np.newaxis]

# Perform element-wise exponentiation
InfectionEvent = np.exp(-bn_values_expanded * t_b_values)
RecoveryEvent = np.exp(-(dn_values_expanded)*t_d_values)

#
TB_probability = []
TB_probability_sum = []
TD_probability = []
TD_probability_sum = []

for i in range(3):
    # compute the probability of infection and recovery (t_b_values and
t_d_values)
    sumTD_values = np.sum(t_b_values[i])
    sumTB_values = np.sum(t_d_values[i])
    TB_probability_sum.append(sumTB_values)
    TD_probability_sum.append(sumTD_values)

    counts_t_b = np.unique(t_b_values[i], return_counts=True)
    counts_t_d = np.unique(t_d_values[i], return_counts=True)
    # now sort the tuples by their counts descending
    sorted_TB = sorted(list(zip(*counts_t_b)), key=lambda x: x[1], reverse=
True)
    sorted_TD = sorted(list(zip(*counts_t_d)), key=lambda x: x[1], reverse=
True)

    TB_probability.append(sorted_TB)
    TD_probability.append(sorted_TD)


# %%
print(len(TB_probability))
print(len(TB_probability_sum))
print(InfectionEvent.shape)
print(RecoveryEvent.shape)

# %%
# plot the six subplots
fig, axs = plt.subplots(3, 2, figsize=(10, 13))
fig.suptitle('Infection and recovery events for different parameter
combinations')
for i in range(3):

    distribution_t_b = np.array(np.unique(t_b_values[i], return_counts=True)).T
    distribution_t_b[:, 1] /= np.sum(distribution_t_b[:, 1])

    distribution_t_d = np.array(np.unique(t_d_values[i], return_counts=True)).T
    distribution_t_d[:, 1] /= np.sum(distribution_t_d[:, 1])
    axs[i, 0].semilogy(t_b_values[i],  InfectionEvent[i]  ,label='b_n = ' +
str(bn_values[i]))
    axs[i, 0].semilogy(distribution_t_b[:,0], (distribution_t_b[:,1]))
    axs[i, 0].set_title('Infection events for b_n = ' + str(bn_values[i]) + '
and d_n = ' + str(dn_values[i]))
    axs[i, 0].set_xlabel('Time')
    axs[i, 0].set_ylabel('Probability($t_b$)')
    axs[i, 0].legend()

    axs[i, 1].semilogy(t_d_values[i],  RecoveryEvent[i] ,label='d_n = ' +
str(dn_values[i]))
    axs[i, 1].semilogy(distribution_t_d[:,0], (distribution_t_d[:,1]))
    axs[i, 1].set_title('Recovery events for b_n = ' + str(bn_values[i]) + '
and d_n = ' + str(dn_values[i]))
    axs[i, 1].set_xlabel('Time')
    axs[i, 1].set_ylabel('Probability($t_d$)')
    axs[i, 1].legend()

plt.tight_layout()
plt.subplots_adjust(top=0.9)

plt.savefig('infection_recovery_events.png')
plt.show()




```

```python
# %%
# find average time of infection and recovery
for i in range(3):
    print(f"mean time recovery(tb): \t{np.round(np.mean(np.array(t_b_values[i])), 2)} s")
    print(f"mean time infection(td): \t{np.round(np.mean(np.array(t_d_values[i])), 2)} s")



# %% [markdown]
# D) Population distribution at different times in the stochastic model
#



```