Mathmatical Modelling Exam

I. Adult-children discrete time model

a. Discrete function for the next time step

The adult A_n and children population can be described by eq. 1.

$$\begin{pmatrix} A_{n+1} \\ C_{n+1} \end{pmatrix} = \begin{pmatrix} max(C_n + A_n(1-dA_n),0) \\ max(rA_n,0) \end{pmatrix} \tag{1}$$

b. Equilibria

$$A_* = C_* + A_*(1 - d * A_*) \tag{2}$$

$$C_* = r * A_* \tag{3}$$

Inserting eq. 3 in eq. 2.:

$$0 = r * A_* - d * A_*^2 \tag{4}$$

The equilibria for (A, C) can thus be determined to $(A_{*1}, C_{*1}) = (0, 0)$ and $A_{*2}, C_{*2} = (\frac{r}{d}, \frac{r^2}{d})$.

c. Stability

The eigenvalues of the Jacobian (eq. 5) yield information about the stability of the equilibria.

$$J = \begin{pmatrix} 1-2dA_n & 1 \\ r & 0 \end{pmatrix} \tag{5}$$

Near (0,0) we thus receive $\lambda_1 = 0.5(1-\sqrt{4r+1}$ and $\lambda_2 = 0.5(1+\sqrt{4r+1}$. As $r > 0$, we have one eigenvalue that is greater than zero and thus the equilirbium is unstable.

Next, we determine the eigenvalues of the Jacobian (eq. 5) near A_{*2} to be $\lambda_1 = \frac{1}{2}[1-2r-\sqrt{4r^2+1}]$ and $\lambda_2 = \frac{1}{2}[1-2r+\sqrt{4r^2+1}]$. It follows that $\lambda_1 \leq 0$ for $r \geq 0$ and $\lambda_2 > 0$. Thus, A_{*2} is not a stable equilibrium.

d. 2-periodic points

The two periodic points are described by:

$$\begin{pmatrix} A_{n+2} \\ C_{n+2} \end{pmatrix} = \begin{pmatrix} max(C_{n+1} + A_{n+1}(1-dA_{n+1}),0) \\ max(rA_{n+1},0) \end{pmatrix} \tag{6}$$

We will not write down the maximum function in the following calculations to improve readibility.

$$\begin{pmatrix} A_{n+2} \\ C_{n+2} \end{pmatrix} = \begin{pmatrix} rA_n + A_{n+1}(1-dA_{n+1}) \\ rA_{n+1} \end{pmatrix} \tag{7}$$

$$\begin{pmatrix} A_{n+2} \\ C_{n+2} \end{pmatrix} = \begin{pmatrix} rA_n + A_{n+1}(1-dA_{n+1}) \\ r[C_n + A_n(1-dA_n)] \end{pmatrix} \tag{8}$$

Applying the conditions of the two harmonic points:

$$\begin{pmatrix} A_{n+2} \\ C_{n+2} \end{pmatrix} = \begin{pmatrix} A_n \\ C_n \end{pmatrix} = \begin{pmatrix} A_* \\ C_* \end{pmatrix} \tag{9}$$

leads to:

$$C_* = r\frac{A_*(1-dA_*}{1-r}) \tag{10}$$

which can be used to solve eq. 11.

$$A_* = r \cdot A_* + [C_* + A_* \cdot (1 - d \cdot A_*)](1 - d \cdot [C_* + A_*(1 - d \cdot A_*)] \tag{11}$$

As the equations turns out to be quite complex, we use the *sympy* toolbox to solve the equation.

```
A, r, d = symbols('A r d')

C2 = r * (A*(1-d*A)/(1-r))
A1 = C2 + A * (1-d*A)

expr = r * A + A1 *(1-d*A1)
sol = solve(expr, A)
```

The resulting solutions are displayed in the appendix. While the complexity of the solutions prevents direct interpretation, numerical analysis can be applied in order to obtain insight into their meaning. In our deterministic population model, the four solutions should yield constant or oscillating solutions for every value pair of r and d. Thus, we chose two arbitrary values and modeled the resulting populations with the following code (also more readable on `https://github.com/gebhardleon/MEC658B_ModellingLivingSystems`):

```
a = []
c = []
r_val =1.1
d_val = 1
a_0 = sol[3].as_real_imag()[0].evalf(subs={r:r_val, d:d_val})
c_0 = C2.evalf(subs={r:r_val, d:d_val, A:a_0})
a.append(a_0)
c.append(c_0)

n = 100
for i in range(n):
    a_n = max(c[-1] + a[-1]* (1-d_val * a[-1]),0)
    a.append(a_n)
    c.append(a[-2]*r_val)

plt.plot(a, label='a')
plt.plot(c, label='c')
plt.legend()
plt.show()
```

The results are displayed in fig. 2. Fig. 1a describes the trivial solution of $A_0 = C_0 = 0$. Fig. 1b and fig. 1c describe non-sense solutions with intial values below zero. Finally, fig. 1d describes an oscillatory diverging population.

II. Continous age population

a. Discrete population model

Next, we want to model a population with K discrete ages using an age-specific death rate $d : f(a)$ and reproduction rate $r : f(a)$. The difference between the discrete ages ∂a is equal to the discrete time steps ∂t. Thus, the population of the next time step $n+1$ at each age k can be described as:

$$P_k^{n+1} = P_{k-1}^{n}(1 - P_{k-1}^{n}), k \neq 0 \tag{12}$$

for all ages, expect $k = 0$. The newborns are described by eq. 13.

$$P_0^{n+1} = \Sigma_{i=0}^{K} P_i^n * r_i \tag{13}$$

b. Continous population model

The number of people "outgrowing" the age a due to a change in time ∂t is equal to the sum of the difference between the population $P(a,t)$ at age a and time t and the population at $P(a + \partial a, t)$ and the dying population $d(a) * P(a,t)$ at this age and time. This equality is expressed in the eq. 14.

$$-[P(a, t + \partial t) - P] = P(a + \partial a, t) - P + d \cdot P(a,t) \tag{14}$$

Working at the limit $\partial t = \partial a \to 0$, eq. 14 converges towards eq. 15.

$$-\frac{\partial P}{\partial t} = \frac{\partial P}{\partial a} + d(a) \cdot P \tag{15}$$

The population of newborns can then be modeled at every time point as:

$$P(0,t) = \int_0^{\inf} P(a,t) \cdot r(a)\, da \tag{16}$$

and the initial population is described by eq. 17

$$P(a,0) = f(a) \tag{17}$$

c. Method of characteristics

We can transform the linear PDE (eq. 18)

$$\frac{\partial P}{\partial t} + \frac{\partial P}{\partial a} = -d(a) \cdot P \tag{18}$$

into an ODE along the appropriate curve of form eq. 19.

$$\frac{d}{dt}[P(a(t),t)] = F(P, a(t), t) \tag{19}$$

By applying the chain rule, we find

$$\frac{d}{dt}[P(a(t),t] = \frac{\partial P}{\partial a}\frac{\partial a}{\partial t} + \frac{\partial P}{\partial t}\frac{\partial t}{\partial t} \tag{20}$$

as evidently $\frac{\partial a}{\partial t} = t$ and $\frac{\partial t}{\partial t} = 1$ we find

$$\frac{d}{dt}[P(a(t),t)] = t\frac{\partial P}{\partial a} + \frac{\partial P}{\partial t} = -d(a) \cdot P \tag{21}$$

which can be solved by solving a system of ODE's (eq. 22 and eq. 23).

$$\frac{da}{dt} = t, a(0) = a_0 \tag{22}$$

$$\frac{dP}{dt} = -d(a) \cdot P, P(0) = P(a_0, 0) \tag{23}$$

Eq. 22 results in $a = t + a_0$ and eq. 23 in $P(a(t),t) = P(t + a_0, t)$. Thus, the characteristic function we were looking for is $X(a,t) = t + a_0$ using a_0 as a parameter to select the line on which we solve the ODE.

We can now find a solution for the ODE by integrating eq. 23:

$$\int_{P(a_0,0)}^{P(a_0+t,t)} \frac{dP}{P} = -\int_0^t d(a_0 + t)dt \tag{24}$$

yields eq. 25, where $g(t) = \int_0^t d(a_0 + t)dt$.

$$\log \frac{P(a_0 + t, t)}{P(a_0, 0)} = -g(t) \tag{25}$$

Thus, the solution to P(X(a,t),t) is:

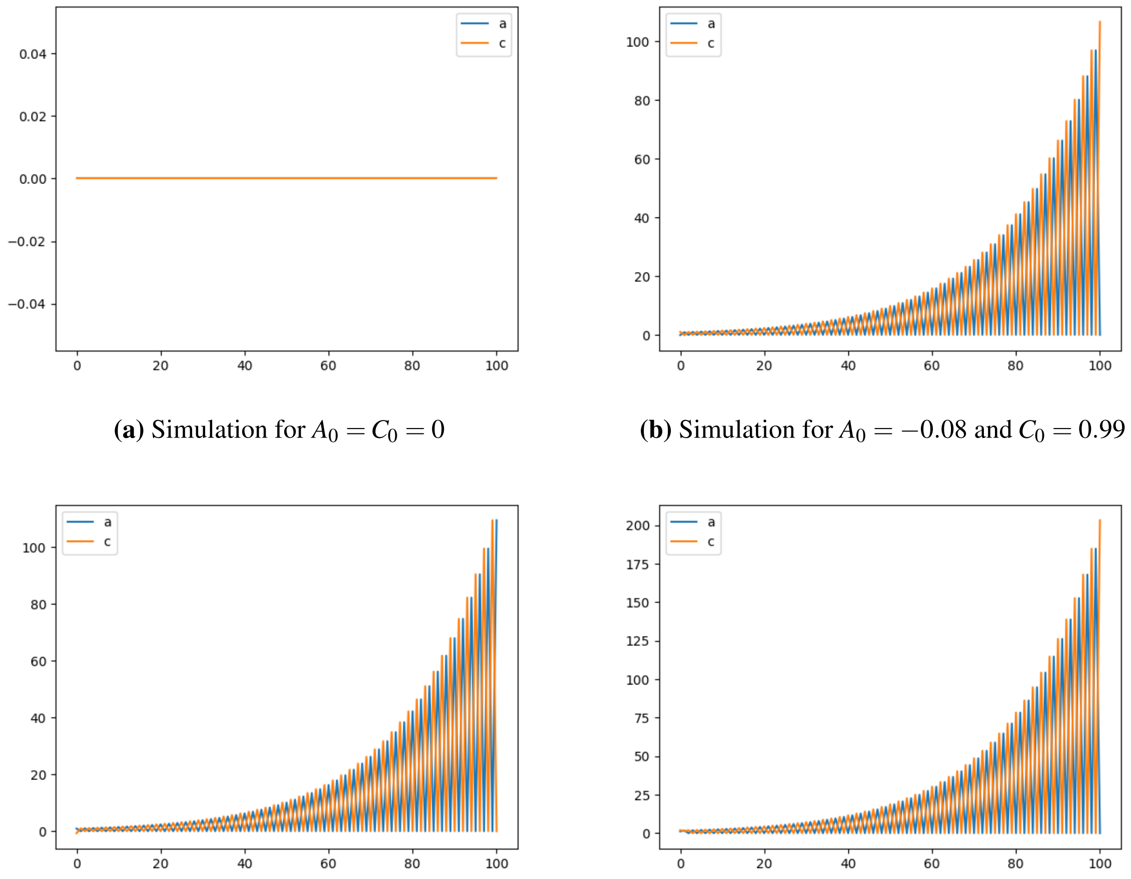
$$P(a_0 + t, t) = P(a_0, 0) \cdot e^{-g(t)} \tag{26}$$

d. Uniqueness of the solution

The uniqueness of a solution can be shown using the Lipschitz criterion (eq. 27), where L denotes the Lipschitz constant.

$$|f(x_2) - f(x_1)| L \cdot |x_2 - x_1| \tag{27}$$

For $|x_2 - x_1| \to 0$ the Lipschitz constant is equal to the partial derivative of the function f or the norm of Jacobian for multidimensional functions.



(a) Simulation for $A_0 = C_0 = 0$

(b) Simulation for $A_0 = -0.08$ and $C_0 = 0.99$

(c) Simulation for $A_0 = 0.93$ and $C_0 = -0.69$

(d) Simulation for $A_0 = 1.15$ and $C_0 = 1.90$

Fig. 1: Numeric simulation of the population sizes using initial populations sizes predicted to result in two-periodic points.

Derivation of eq. 26 using the chain rule leads to eq. 28.

$$\frac{dP}{dt} = -P(a_0,0)\frac{dg(t)}{dt}\cdot e^{-g(t)} \tag{28}$$

The difficulty in determining the Lipschitz constant arises from the unknown function for the death rate $d(t)$ and, correspondingly, the unknown function of its integral $g(t)$. We can still prove the uniqueness of the solution, by investigating the bounds of the derivative. $P(a_0,0)$ being a constant, the existence of the Lipschitz constant depends on the bonds for $\frac{dg(t)}{dt}\cdot e^{-g(t)}$ which is dominated by the exponential term for all $g0$. It follows from $d(t) > 0$ that the integral is also positive. Thus, the derivative is bounded for all positive death rate functions.

e. Numerical scheme

1. Integration of ODE

We integrate eq. 23 from t^n to t^{n+1}.

$$\int_{P^{t^n}}^{P^{n+1}} \frac{dP}{P} = -\int_{t^n}^{t^{n+1}} d(a_i) \tag{29}$$

$$\ln\frac{P(t^{n+1},a_i)}{P(t^n,a_i)} = -\int_{t^n}^{t^{n+1}} d(a_i) \tag{30}$$

2. Approximation of the integral

We apply the approximation, that the changes of $d(a_i)$ between t^{n+1} and t^n are small. We have encountered this scheme previously when approximating the area under a curve to numerically solve integrals with the explicit Euler method.

$$\ln\frac{P(t^{n+1},a_i)}{P(t^n,a_i)} = -[t^{n+1}-t^n]\cdot d(a_i) \tag{31}$$

3. Boundary value P_0^n

Using the explicit Euler approximation, we can determine the boundary value as described in eq. 32.

$$P_0^n = \Sigma_{i=0}^K P_i^{n-1} \cdot r_i \tag{32}$$

III. Dynamics of a transcription factor in a cell

a. Self-activating transcription factor

The change in concentration of a self-activating transcription factor is influenced by its activation $\frac{\alpha c}{1+\beta c}$ and its degradation δc which is summarised in eq. 33. For $c \to 0$ the activation is proportional to the concentration, and for $c \to \inf$ the activation is saturated and thus no longer influenced by the concentration. Here, α and β are constants describing the saturation of the activation, and δ describes the degradation constant.

$$\frac{dc}{dt} = \frac{\alpha c}{1+\beta c} - \delta c \tag{33}$$

b. Limited degradation by concentration of another enzyme

We can modify the degradation term in eq. 33 in order to account for the influence of another enzyme on the degradation of the enzyme either by adding a direct proportionality of the enzyme concentration to the degradation term (eq. 34) or by again incorporating a limited proportionality as in the activation term (eq. 35).

$$\frac{dc}{dt} = \frac{\alpha c}{1+\beta c} - \delta e c \tag{34}$$

$$\frac{dc}{dt} = \frac{\alpha c}{1+\beta c} - \delta \frac{\eta e}{1+\phi e} c \tag{35}$$

c. Equilibria

We now assume the concentration to be governed by eq. 36.

$$\frac{dc}{dt} = \frac{c^2}{1+c^2} - \delta \frac{c}{1+c} = f(c,t) \tag{36}$$

We find the equilibria by setting $\frac{dc}{dt} = 0$ yielding us the three equilibria $c_1 = 0$, $c_{2,3} = \frac{-1 \pm \sqrt{1+4\delta-4\delta^2}}{2-2\delta}$. To investigate their stability, we compute the derivative of $f(c,t)$ in respect to the concentration.

$$\frac{df(c,t)}{dc} = \frac{2c}{(c^2+1)^2} - \frac{\delta}{(c+1)^2} \tag{37}$$

We calculate the stability of the function based on values of δ as displayed in tab. 1. We find four sta-

δ	$\frac{df(c_1)}{dc}$	$\frac{df(c_2)}{dc}$	$\frac{df(c_3)}{dc}$
0.5	🟩 -0.5	🟥 0.35	-0.35 neg. conc.
1	🟩 -1	zero div.	zero div.
$11/10$	🟩 -1.1	🟥 0.16	🟩 $-8.7 \cdot 10^{-3}$

Table 1: Stability analysis of the equilibria for eq. 36 by evaluating eq. 37 Red colours indicate unstable equilibria, green colour stable equilibria and uncolured cells indicate either a division by zero error or nonphysical results due to a negative concentration.

ble equilibria and two unstable equilibria based on the sign of eq. 37. $c_3(\delta = 0.5)$ results in a negative concentration. Thus, even though the equilbrium is stable, we did not count it because negative concentrations have no physical meaning in this context. Furthermore, the roots $c_{2/3}$ are undefined in $\delta = 1$ and thus the behaviour of c near them needs further investigation.

d. Numerical analysis

We perform this further investigation of $c(t, \delta = 1)$ numerically by approximating the integral of eq. 36 using an explicit Euler approach.

$$\frac{c_{n+1} - c_n}{\Delta t} = \frac{c_n^2}{1+c_n^2} - \frac{c_n}{1+c_n} \tag{38}$$

The roots of this equation are in $c_n = 0$ and $c_n = 1$. Having shown the stability of $c_n = 0$ analytically, we have particular interest in the behaviour of the concentration around $c_n = 1$. Thus, we determine $c_{n+1} = f(c_0, n * \Delta t)$ for three different initial concentrations: $1 - \varepsilon, 1$ and $1 + \varepsilon$.

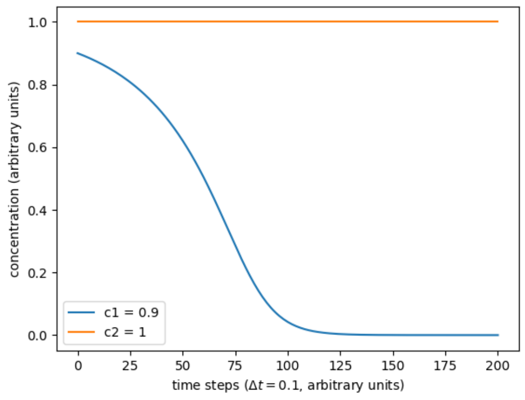
$$c_{n+1} = c_n + \Delta t [\frac{c_n^2}{1+c_n^2} - 1 \frac{c_n}{1+c_n}] \tag{39}$$

The code used for the analysis can be found in the Appendix (section b) or on GitHub: `https://github.com/gebhardleon/MEC658B_ModellingLivingSystems`.

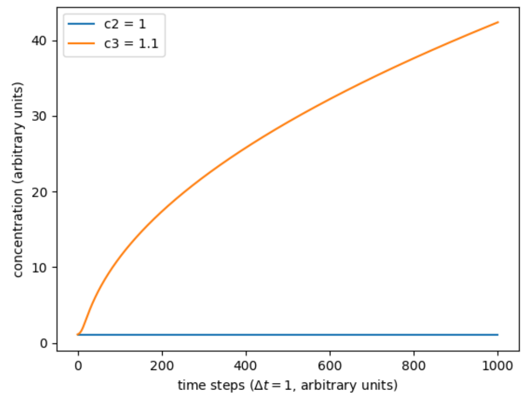
Fig. 2a demonstrates that for concentrations above one, the concentration diverges, while it converges to zero for concentrations below one. Thus, we have finished our previous analysis and shown, that the root $c_n = 1$ is unstable.

IV. Dynamics of a transcription factor in a tissue

We describe the concentration change over space and time using a non-linear parabolic equa-



(a) Numerical solution of eq. 39 for 200 time steps using $\Delta t = 0.1$.



(b) Numerical solution of eq. 39 for 1000 time steps using $\Delta t = 1$.

Fig. 2: Numerical solution of eq. 39 for different initial concentrations c_0 using $\delta = 1$.

tion, which has strong similarities with reaction-diffusion or heat-transfer equations.

$$\frac{\partial c}{\partial t} = \frac{c^2}{1+c^2} - \frac{c}{1+c^2} + dx\frac{\partial c^2}{\partial x^2} \tag{40}$$

Here, the change of concentration $\frac{\partial c}{\partial t}$ in time is equal to the self-activated production of the protein $\frac{c^2}{1+c^2}$ and the diffusion-based flux $dx\frac{\partial c^2}{\partial x^2}$ minus the degradation $\frac{c}{1+c^2}$. This description of the concentration changes is relevant on all scales that are diffusion-dominated and thus have no underlying convection. Furthermore, this model assumes homogeneous diffusion, so it is valid at the tissue scale, where the diffusive resistance caused by individual cells (membranes) can be averaged out.

a. Numerical solution of eq. 40

To numerically solve eq. 40 we approximate the spatial derivative as:

$$\frac{\partial c^2}{\partial x^2} = \frac{c_n^{l+\Delta l} - 2c_n^l + c_n^{l-\Delta l}}{\Delta l^2} \tag{41}$$

Thus, we receive a new determining term for c_{n+1}:

$$c_{n+1} = c_n + \Delta t[\frac{c_n^2}{1+c_n^2} - \frac{c_n}{1+c_n} + \frac{c_{n+1} - 2c_n + c_{n-1}}{\delta L^2}] \tag{42}$$

Solving this system iteratively with a standard explicit euler approach can be unstable depending on the diffusion coefficient d, step size ΔL, and time steps Δt. To demonstrate this, we solved eq. 42 twice with the same boundary conditions $c(0,t) = 0, \partial_x c(L,t) = 0$, initial condition $c(x,0) = x * 0.01$, time and space step size but different diffusion coefficients. As displayed in figure 3 using the higher diffusion coefficient with the same step size results in a diverging, fluctuating solution that loses its physical meaning.

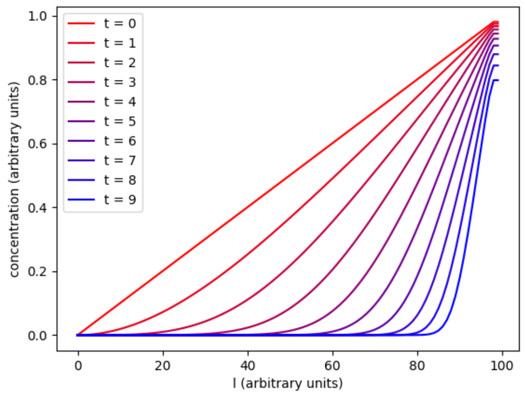
The lack of convergence is likely caused by floating-point errors during the computation of eq. 41 as they errors are amplified by the division with ^{2}. The effect of floating point errors can be minised by choosing smaller time steps Δt as this decreases the weight of the errors and prevents their propagation. However, the size of the time steps is inversely proportional to the computation time, making an appropriate choice of this step size essential. We overcome this challenge by choosing a dynamic time step according to the Courant-Friedrichs-Lewy condition (eq. 43) which is bound by the maximum and minimum step sizes $t_{max} = 10^{-3}; t_{min} = 10^{-7}$. Generally, better convergence could still be achieved by using more stable algorithms, such as runge-kutta.

The code is again provided in the appendix in section c or on Github: `https://github.com/gebhardleon/MEC658B_ModellingLivingSystems`.

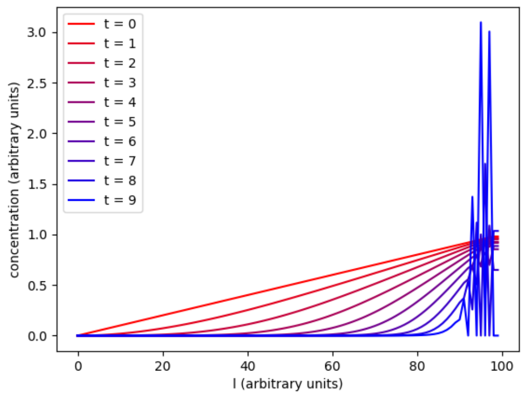
$$\Delta t \leq \frac{\Delta L}{2 \cdot d \cdot max(\frac{c_n^{i+1} - c_n^i}{\Delta L})} \tag{43}$$

b. Investigation of behaviour for different lengths and diffusion coefficients

Having obtained a numerical solution for the concentration profile, we can assess how different sys-



(a) Numerical solution of eq. 42 for a diffusion coefficient of $d = 0.1$, time step $= 1$ and step size $\Delta l = 1$ results in a convergence towards zero on all grid points.



(b) Numerical solution of eq. 42 for a diffusion coefficient of $d = 1.1$, time step $= 1$ and step size $\Delta l = 1$ results in a divergence of the solving algorithm.

Fig. 3: Numerical solution of eq. 42 for diffusion coefficients on $0L100$ for ten time steps with $\delta L = \delta t = 1$. The initial concentration is set to $c(x,0) = 0.01x$.

tem sizes (L) and diffusion coefficients (d) affect the time required by the system to reach an equilibrium state using $c(x,0) = tanh(x)$ as initial concentration.

Fig. 4a indicates a linear dependence of the computation time on the system size. This demonstrates that for all investigated parameters, the system's behavior is dominated by the diffusion and not the production or degradation of the gene.

The curves resulting from a display of the logarithmic equilibrium time over the logarithmic diffusion coefficient also demonstrate quasi-linear behavior, suggesting that they might partially be explained by an equation of the form $t_eq = d^k + const$ where k indicates the slope of the log-log plot.

d \ L	10	30	50	100
1	15.47	36.91	57.56	108.46
10	7.35	15.82	22.82	39.53
100	1.73	6.10	9.09	15.08
1000	0.21	1.12	2.13	4.47

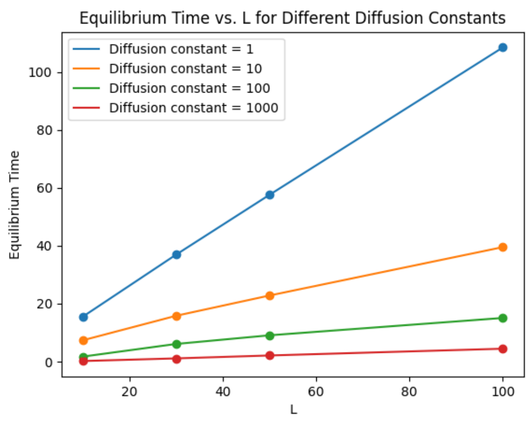
Table 2: Time required by the system until the maximum change in concentration between two time steps is below 10^{-5} for 10 consecutive time steps.

However, calculating the slopes at each length and diffusion coefficient demonstrates, that the slope is always the highest between $L = 100$ and $L = 1000$, indicating that this law is only applicable to small systems. Considering that the obtained steady state was mainly reached through a flux of the protein in the $c(0,t) = 0$ boundary, this relationship is logical, as the diffusive influence on the concentration profile remains constant with an increasing system, while the production and degradation terms scale linearly with the system size.

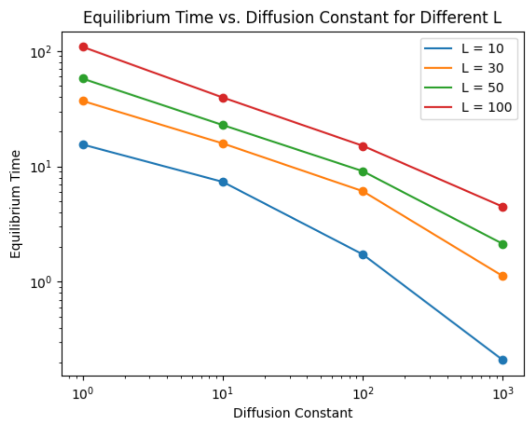
V. Appendix

a. Results excersize 1.4

2-periodic points for A: Solution 1:
0
Solution 2:
-(-3*(2 - r)/d**2 + 4/d**2)/(3*(sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)) - (sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)/3 + 2/(3*d),
Solution 3:
-(-3*(2 - r)/d**2 + 4/d**2)/(3*(-1/2 - sqrt(3)*I/2)*(sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)) - (-1/2 - sqrt(3)*I/2)*(sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)/3 + 2/(3*d),
Solution 4:
-(-3*(2 - r)/d**2 + 4/d**2)/(3*(-1/2 +



(a) Dimensionless time until the system reaches equilibrium over the size of the system for different diffusion constants.



(b) Double logarithmic plot of the equilibrium time over the diffusion constant for differenz sizes of the system.

Fig. 4: Dependance of the equilibrium time on the size and diffusion coefficient of the system.

sqrt(3)*I/2)*(sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)) - (-1/2 + sqrt(3)*I/2)*(sqrt(-4*(-3*(2 - r)/d**2 + 4/d**2)**3 + (18*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/d**3 - 16/d**3)**2)/2 + 9*(2 - r)/d**3 + 27*(-r**3 + 2*r**2 - 1)/(2*d**3) - 8/d**3)**(1/3)/3 + 2/(3*d)]

b. Code for Excersize 2

Please find the codes in readable size under this link::
https://github.com/gebhardleon/MEC658B_ModellingLivingSystems

```
import matplotlib.pyplot as plt
from sympy import symbols, solve, evalf

c, dt = symbols('c dt')
c_new = c +dt *(c ** 2 / (1 + c ** 2) -  c / (1 + c))

c_0 = [0.9,1, 1.1]
c1 = [c_0[0]]
c2 = [c_0[1]]
c3 = [c_0[2]]
t = 1
for i in range(1000):

    c2.append(c_new.evalf(subs={c:c2[i], dt:t}))
    c3.append(c_new.evalf(subs={c:c3[i], dt:t}))
plt.plot(c2, label='c2')
plt.plot(c3, label='c3')

plt.xlabel('time steps ($\Delta t = 1$, arbitrary units)')
plt.ylabel('concentration (arbitrary units)')
c2 = [c_0[1]]
plt.legend()
plt.savefig('Exc2_sol4_diverge.png')
plt.show()
t = 0.1
for i in range(200):
    c1.append(c_new.evalf(subs={c:c1[i], dt:t}))
    c2.append(c_new.evalf(subs={c:c2[i], dt:t}))

plt.plot(c1, label='c1')
plt.plot(c2, label='c2')
plt.xlabel('time steps ($\Delta t = 0.1$, arbitrary units)')
plt.ylabel('concentration (arbitrary units)')
plt.legend()
plt.savefig('Exc2_sol4.png')
plt.show()
```

c. Explicit euler with CWL condition

Please find the codes in readable size under this link::
https://github.com/gebhardleon/MEC658B_ModellingLivingSystems

```
import matplotlib.pyplot as plt
import numpy as np
from sympy import symbols

time_steps = 100000
delta_l = 0.5
L_vec = [10, 30, 50, 100]
diff_vec = [100, 1000]
conv_time = []
for diff in diff_vec:
    for L  in L_vec:
        conv_time.append(0)
        # Set initial time step
        delta_t_max = 0.001
        delta_t_min = 0.000001

        # Set convergence criteria
```

```
convergence_threshold = 1e-5  # Change in concentration threshold
consecutive_steps = 10  # Number of consecutive steps below threshold to consider quasi-stationary

c_0 = np.array([np.tanh(i * delta_l) for i in range(round(L / delta_l))])
c_0[0] = 0
c_0[-1] = c_0[-2]
c_n = np.array([c_0])

c, dt, ddcxx = symbols('c dt ddcxx')
c_new = c + dt * (c ** 2 / (1 + c ** 2) - c / (1 + c) + diff * ddcxx)

# Initialize convergence tracking variables
consecutive_converged_steps = 0
last_concentration = c_n[-1]

for i in range(time_steps):
    c_temp = np.zeros_like(c_n[-1])

    # Calculate CFL condition
    max_gradient = np.max(np.abs(np.gradient(c_n[-1], delta_l)))
    delta_t = min(delta_t_max, delta_l ** 2 / (2 * diff * max_gradient))
    delta_t = max(delta_t, delta_t_min)
    conv_time[-1] = conv_time[-1] + delta_t
    if i/time_steps > 0.1:
        print(f'Length {L} and Diff {diff} did not converge at time  {conv_time[-1]}.')
    elif i/time_steps > 0.3:
        print(f'Length {L} and Diff {diff} did not converge at time  {conv_time[-1]}.')
    elif i/time_steps > 0.5:
        print(f'Length {L} and Diff {diff} did not converge at time  {conv_time[-1]}.')
    for j in range(1, len(c_n[-1]) - 1):  # Skipping first and last elements
        # Calculate diffusion term
        diffusion_term = c_n[-1][j + 1] / delta_l ** 2 - 2 * c_n[-1][j] / delta_l ** 2 + c_n[-1][j - 1] / delta_l ** 2
        # Evaluate new concentration using the symbolic expression
        c_temp[j] = c_new.evalf(subs={c: c_n[-1][j], dt: delta_t, ddcxx: diffusion_term}, chop=True)

    # Set boundary conditions
    c_temp[0] = 0
    c_temp[-1] = c_temp[-2]

    # Check for convergence
    change_in_concentration = np.max(np.abs(c_temp - last_concentration))
    if change_in_concentration < convergence_threshold:
        consecutive_converged_steps += 1
        if consecutive_converged_steps >= consecutive_steps:
            print(f"Length {L} and Diff {diff} Converged at time  {conv_time[-1]}.")
            break
    else:
        consecutive_converged_steps = 0


    # Update last concentration for next iteration
    last_concentration = c_temp.copy()

    c_n = np.append(c_n, [c_temp], axis=0)

# Plot every nth line in a different color on a labeled plot
import matplotlib.colors as mcolors

n = 100
for i in range(0, len(c_n), n):
    # Calculate the color based on progress in time
    progress = i / (len(c_n) - 1)  # Progress normalized between 0 and 1
    color = mcolors.to_rgba((1 - progress, 0, progress))  # Fade from red to blue

    plt.plot(np.linspace(0, L, int(L / delta_l)), c_n[i], label='t = ' + str(i * delta_t), color=color)

plt.xlabel('l (arbitrary units)')
plt.ylabel('concentration (arbitrary units)')
pltname = f'Exc4_sol_{L}_{diff}.png'
plt.savefig(pltname)
plt.show()
```

REFERENCES