Computational Biology, Problem set 1, task 2

Amandus Reimer, cid: reimera, Axel Johansson, cid: axejoh

February 2023

This task was solved with Python, using mainly numpy and ddeint as numerical aids. Matplotlib was used for graphics.

Task 1

1 a)

Different dynamics is illustrated below. In the first subplot there are no oscillations. In 2-3 there are damped oscillations and in 4-6 there are stable oscillations.

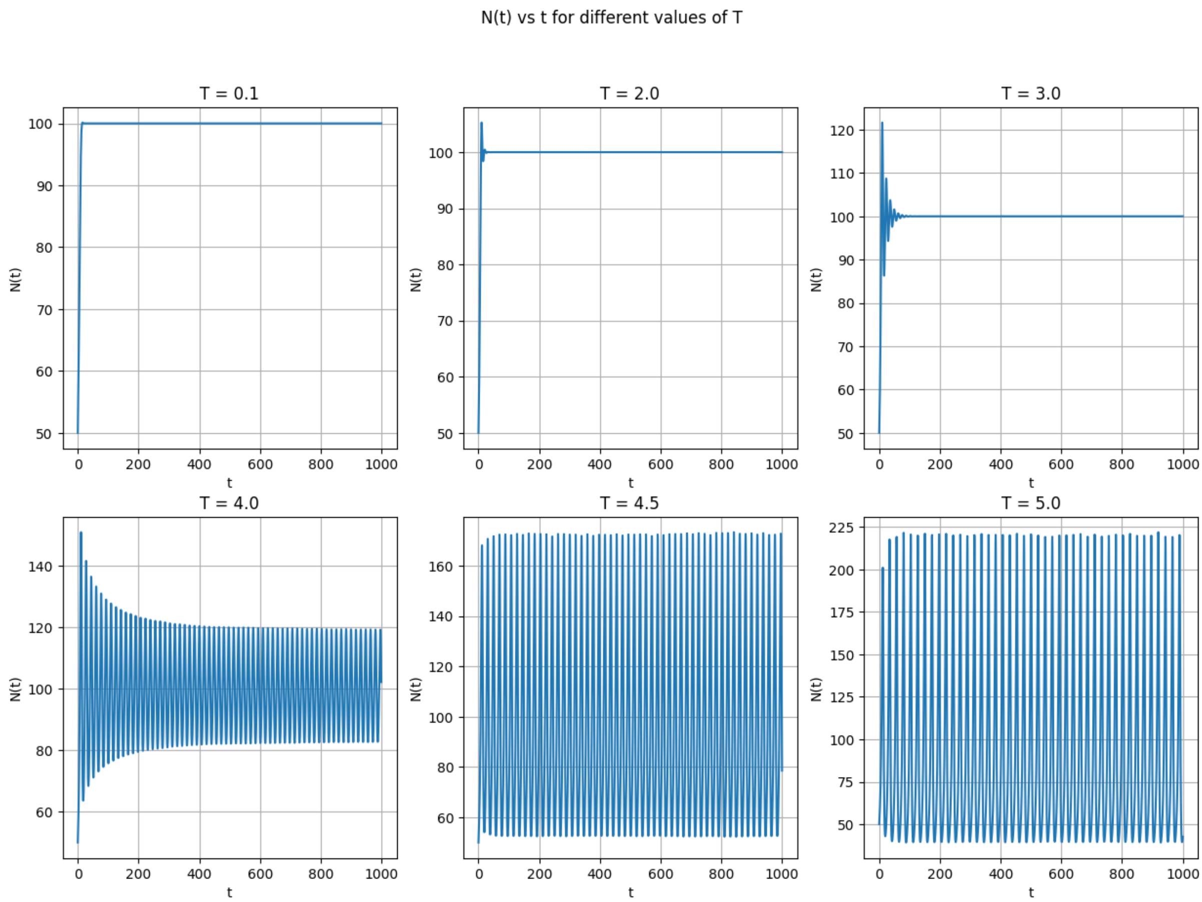


Figure 1: This figure shows different dynamics of the system for different time delays T. For $T = 0.1$ there are no oscillations and when $T \approx 4$ there appears to be a stable limit cycle.

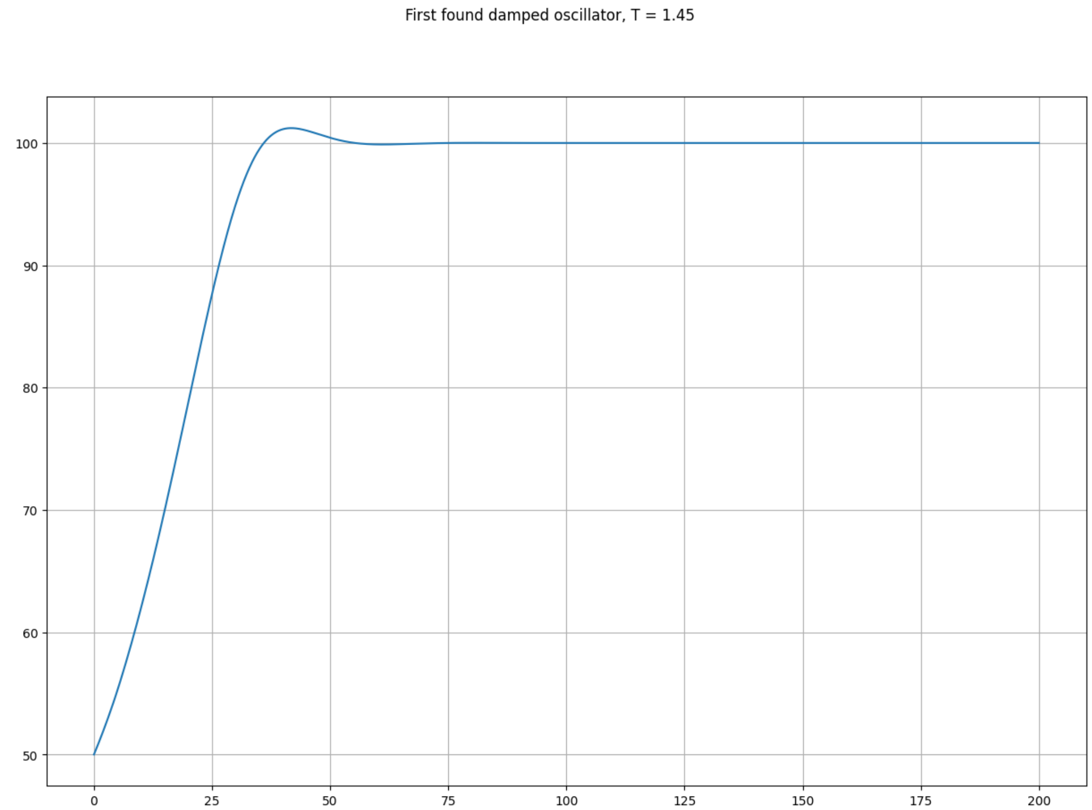


Figure 2: First damped oscillatory behavior at T = T_{damped} = 1.45.

1 b)

We interpreted that oscillatory behaviour occurs when there is an overshoot of the carrying capacity K = 100 with 1 whole individual i.e. when N=101. This occurred at time delay T_{damped} = 1.45 as can be seen in figure 2. However if N is continous this occurs closer to T=1.

1 c)

We found that this oscillatory behaviour occured at T $\approx$ 3.95, since for this value of T we noticed undamped oscillations after the transient behaviour, implying a limit cycle see figure 3.

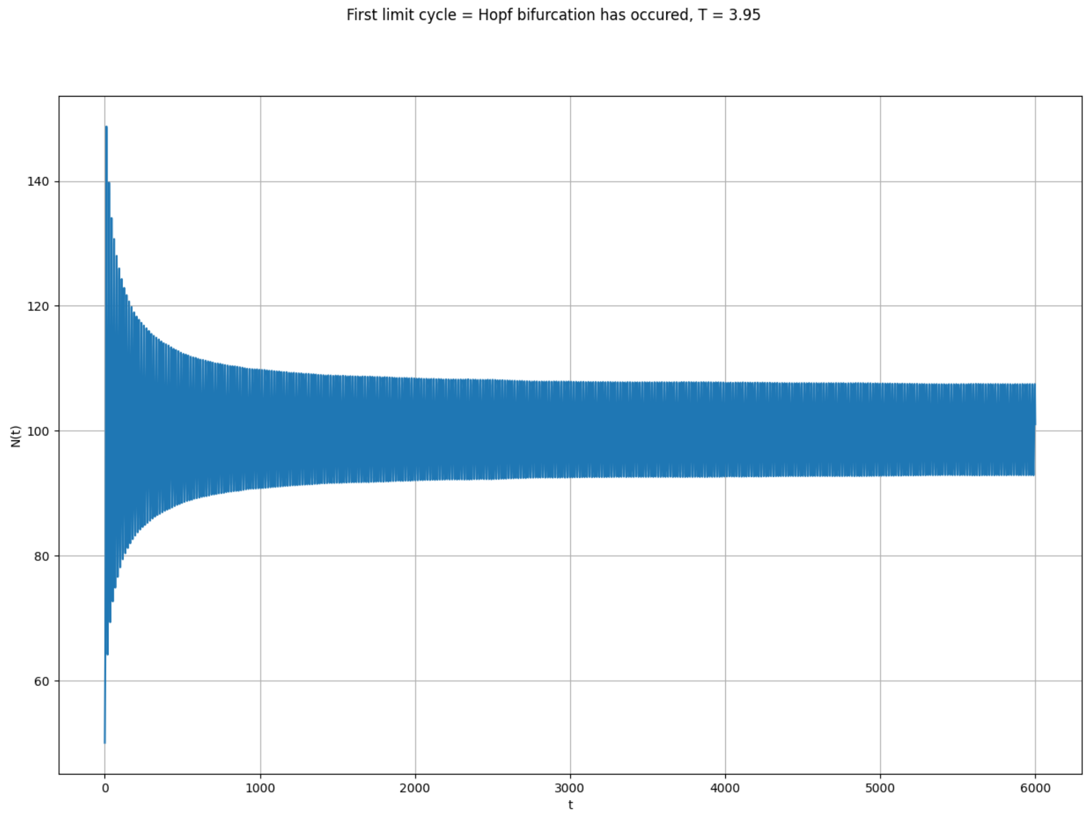


Figure 3: The figure shows our estimated value of the delay, denoted $\mathrm{T_H}$, that shows a stable oscillations. This occurs at around $\mathrm{T_H} = 3.95$.

1 d)

The model is rewritten in dimensionless units using $t = t_0\tau$ and $N(t) = N_0 u(\tau)$. This gives

$$\frac{N_0}{t_0}\frac{d}{d\tau}u(\tau) = rN_0 u(\tau)(1 - \frac{N_0}{K}u(\tau - \frac{T}{t_0}))(\frac{N_0 u(\tau)}{A} - 1)$$

If one let $t_0 = 1/r$ and $N_0 = K$, but we also let $D = T/t_0$, and $A' = A/K$,

$$\frac{d}{d\tau}u(\tau) = u(\tau)(1 - u(\tau - D))(\frac{u(\tau)}{A'} - 1). \tag{1}$$

If one expands this equation around a steadystate with $u(\tau) = u_{\text{steady}} + \eta(\tau)$ where $\eta(\tau) \ll 1$ the equation becomes

$$\frac{d}{d\tau}u(\tau) = (u_{\text{steady}} + \eta(\tau))(1 - u_{\text{steady}} - \eta(\tau - D))\left(\frac{u_{\text{steady}}}{A'} - 1 + \frac{\eta}{A'}\right). \tag{2}$$

It's given that $N^* = K \implies u^* = 1$ and the equation simplifies if we throw away the higher order terms to

$$\frac{d}{d\tau}\eta(\tau) = -\frac{\eta(\tau - D)}{A'} + u(\tau - D) = \frac{A' - 1}{A'}\eta(\tau - D). \tag{3}$$

But from the definition of A' we know that $(A' - 1)/A' = (0.2 - 1)/0.2 = -4$. We are asked to look for solutions on form $\eta(\tau) = \eta_0 e^{\lambda\tau}$ so

$$\eta_0 \lambda e^{\lambda\tau} = -4\eta_0 e^{\lambda(\tau - D)} \implies \lambda = -4e^{-\lambda D} \tag{4}$$

Let $\lambda = \lambda' + i\lambda''$ hence the real and imaginary solutions are

$$\begin{cases} \text{Re}: & \lambda' = -4e^{-\lambda' D}\cos(\lambda'' D) \\ \text{Im}: & \lambda'' = 4e^{-\lambda' D}\sin(\lambda'' D) \end{cases} \tag{5}$$

Looking for solutions when $\lambda' = 0$. Imaginary part gives

$$D = \frac{\arcsin \frac{\lambda''}{4}}{\lambda''} \tag{6}$$

which is inserted in the real part as follows

$$0 = \cos \arcsin \frac{\lambda''}{4} \implies \lambda'' = 4 \tag{7}$$

Then with $\lambda'' = 4$

$$0 = \cos 4D \implies D = \frac{\arccos 0}{4} = 0.393 \tag{8}$$

and since

$$D = rT \implies T_H = 3.93. \tag{9}$$

This is well in line with what we saw in the simuation!

Python code

```
def main(T, endtime):
    A = 20
    K = 100
    r = 0.1
    NO = 50
    ts = np.linspace(0,endtime,endtime*2)
    ns = ddeint(Nprim, initial_history, ts, fargs=(r, K, A, T))

    return ns

def Nprim(N, t, r, K, A, T):
    return r*N(t)*(1-N(t-T)/K)*(N(t)/A-1)

def initial_history(t):
    return 50.

def gen_images(endtime):
    ts_ = np.linspace(0,endtime,endtime*2)
    ns_= np.zeros((6, len(ts_)))
    fig, ax = plt.subplots(2,3, figsize=(15,10))
    for idx, time in enumerate(np.array([0.1, 2, 3, 4, 4.5, 5])):
        ns_[idx] = main(time, endtime)
        ax[idx//3, idx%3].plot(ts_, ns_[idx])
        ax[idx//3, idx%3].set_title('T = {}'.format(time))
        ax[idx//3, idx%3].set_xlabel('t')
        ax[idx//3, idx%3].set_ylabel('N(t)')
        ax[idx//3, idx%3].grid(True)
    fig.suptitle('N(t) vs t for different values of T')
    plt.show()

def findDampedOsc(endtime):
    ts_ = np.linspace(0,endtime,endtime*2)
    ns_= np.zeros(len(ts_))
    fig, ax = plt.subplots(1,1, figsize=(15,10))
```

```
    for idx, time in enumerate(np.arange(0.1, 5, 0.05)):
        ns_ = main(time, endtime)
        if np.max(ns_) >101:
            print('T = {}'.format(time))
            break
    plt.plot(ts_, ns_)
    plt.grid(True)
    fig.suptitle('First found damped oscillator, T = {}'.format(np.round(time, 2)))
    plt.show()


def plotSingleT(T, endtime=200):
    ts_ = np.linspace(0,endtime,endtime*2)
    ns_= main(T, endtime)
    fig, ax = plt.subplots(1,1, figsize=(15,10))
    ax.plot(ts_, ns_)
    ax.set_xlabel('t')
    ax.set_ylabel('N(t)')
    ax.grid(True)
    fig.suptitle('First limit cycle = Hopf bifurcation has occured, T = {}'.format(3.95))
    plt.show()

def findHopf(endtime):
    ts_ = np.linspace(0,endtime,endtime*2)
    ns_= np.zeros(len(ts_))
    fig, ax = plt.subplots(1,1, figsize=(15,10))
    for idx, time in enumerate(np.arange(4, 5, 0.05)):
        ns_ = main(time, endtime)
        if np.max(ns_) >101:
            if len(np.where((np.max(ns_)-0.001< ns_) & (ns_ <0.001+ np.max(ns_)))[0]) > 1:
                print('T = {}'.format(time))
                break

    plt.plot(ts_, ns_)
    plt.xlabel('t')
    plt.ylabel('N(t)')
    plt.grid(True)
    fig.suptitle('First limit cycle = Hopf bifurcation has occured, T ={}' ...
    .format(np.round(time, 2)))
    plt.show()

gen_images(1000)
#plotSingleT(3.95, 6000)
#findDampedOsc()
#findHopf()
```

Task 2

2 a)

The non-negative steady states of model

$$N_{\tau+1} = \frac{(r+1)N_\tau}{1+(\frac{N_\tau}{K})^b} \quad = F(N_\tau), \tag{10}$$

are the values of N_τ at which the population size doesn't change over time i.e. steady states are the solutions of $N_\tau = N_{\tau+1}$. Substituting this in equation (10) gives

$$N_\tau = \frac{(r+1)N_\tau}{1+(\frac{N_\tau}{K})^b} \implies N_\tau\left(1+\left(\frac{N_\tau}{K}\right)^b\right) = (r+1)N_\tau \implies N_\tau\left(1+\left(\frac{N_\tau}{K}\right)^b - (r+1)\right) = 0. \tag{11}$$

So there are two non negative steady states. A trivial where $N_1^* = 0$ and one where

$$\left(1+\left(\frac{N_\tau}{K}\right)^b - (r+1)\right) = 0 \qquad \implies N_2^* = K\sqrt[b]{r}. \tag{12}$$

2 b)

Linear stability analysis is used to determine whether the population size will deviate from the fixpoint in a slightly perturbed state. To perform the linear stability analysis, we consider small deviations from the steady state, represented by $N_\tau = N + \eta(\tau)$. This gives us

$$N^* + \eta_{\tau+1} = f(N^* + \eta_\tau) = f(N^*) + f'(N^*)\eta_\tau. \tag{13}$$

Since $N^* = f(N^*)$ this gives the condition

$$\eta_{\tau+1} = f'(N^*)\eta_\tau \tag{14}$$

when $|f'(N^*)| < 1$ the fixpoint is stable. The derivative of the model (10) at $N^* = 0$,

$$f'(N_1^*) = (1+r) \tag{15}$$

and at $N^* = K\sqrt[b]{r}$

$$f'(N_2^*) = 1 - \frac{br}{1+r}. \tag{16}$$

2 c)

When $|f'(N^*)| < 1$ the steadystate is stable and when the norm surpasses 1 a bifurcation occurs making it unstable. To find where a bifurcation happens we investigate for what parameters it's stable,

$$-1 < 1 - \frac{br}{1+r} < 1. \tag{17}$$

However since $br/(1+r)$ positive $\forall b \geq 1$ and $r > 0$ then $f'(N_2^*) < 1$. Which means a bifurcation occurs only when $f'(N_2^*) < -1$. The problem is stable for,

$$-1 < 1 - \frac{br}{1+r} \implies b < \frac{2(1+r)}{r}. \tag{18}$$

Hence the model exhibits stable oscillations when $b < 2(1+r)/r$ and the bifurcation occurs when $b \geq 2(1+r)/r$ leading to unstable oscillations.

2 d)

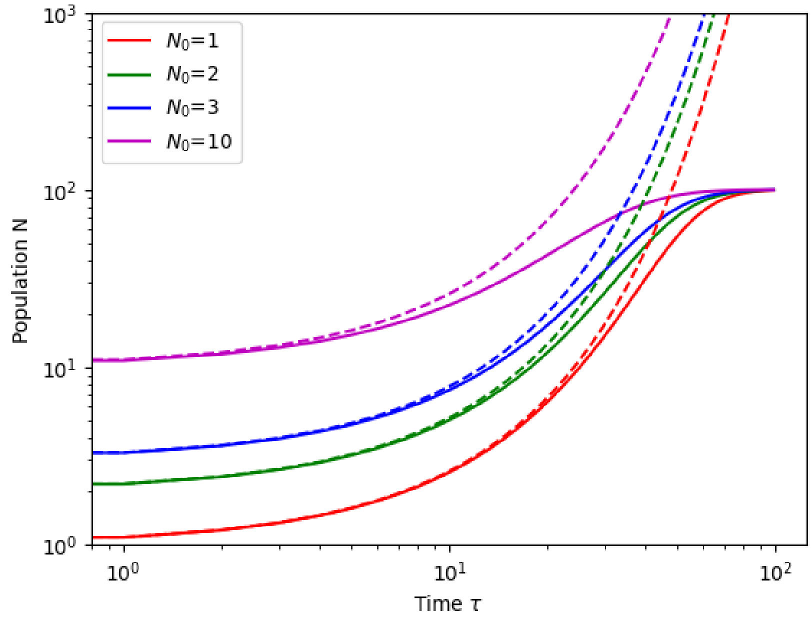


Figure 4: Model around the unstable fixpoint. Dotted lines from linear stability with perturbation. As can be seen the perturbation grow with time.

2 e)

Linear stability approximates well in the beginning but after a few timesteps the difference becomes large. Larger starting populations seem to deviate earlier in absolute terms.

2 f)

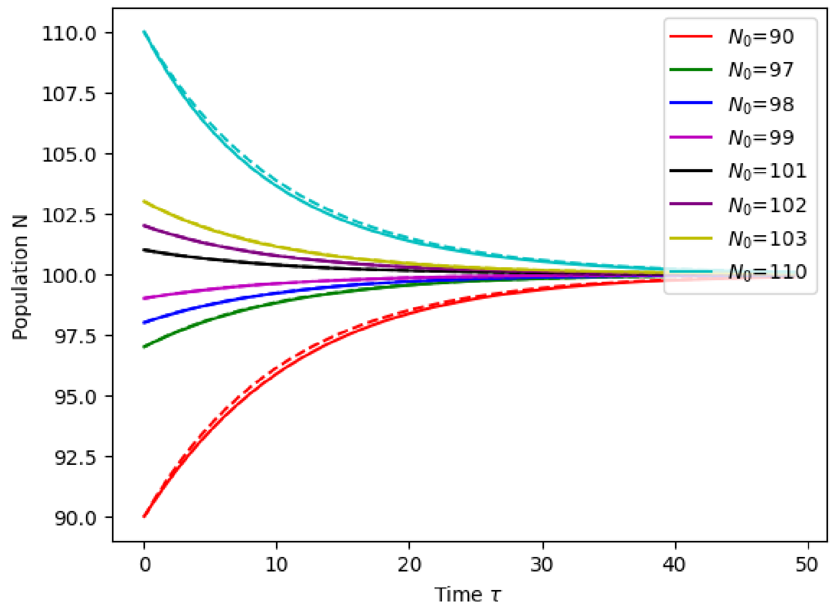


Figure 5: Linear stability around the stable fixpoint for different starting positions. The perturbation doesn't grow with time.

Python code

```
import numpy as np
import matplotlib.pyplot as plt

K = 1e3
r = .1
b = 1
N0 = [1, 2, 3, 10]
time_max = 100
N = np.zeros((len(N0), time_max))
N[:,0] = N0
N_linear = np.copy(N)

def ComputePopulation(N):
    for t in range(time_max-1):
        N[:,t+1] = (r+1)*N[:, t] / (1+(N[:, t]/K)**b)
    return N

def ComputeLinearStabilityPopulation():
    for t in range(time_max-1):
        N_linear[:,t+1] = (1+r) * N_linear[:,t]
    return N_linear

N = ComputePopulation(N)
N_linear = ComputeLinearStabilityPopulation()

color = ['r','g','b','m']
for i, row in enumerate(N):
    plt.loglog(row,c=color[i],label= f'$N_0$={N0[i]}')

for i, row in enumerate(N_linear):
    plt.loglog(row, linestyle = '--',  c=color[i])

plt.xlabel(r"Time $\tau$")
plt.ylabel("Population N")
plt.ylim(1,1000)
plt.legend()
plt.show()

# f)
time_max = 50
N_star_2= K*r**(1/b)
dN0 = np.array([-10, -3, -2, -1, 1, 2, 3, 10])
N = np.zeros((len(dN0), time_max))
N_linear = np.copy(N)

N[:,0] = N_star_2 + dN0
N_linear[:,0] = dN0
```

```
for t in range(time_max - 1):
        N_linear[:, t + 1] = (1+r-r*b)/(1+r) * N_linear[:, t]
        N[:,t+1] = (r+1)*N[:, t] / (1+(N[:, t]/K)**b)
N_linear += 100

colors = ['r','g','b','m','k','purple','y','c']
for i, color in enumerate(colors):
    plt.plot(range(time_max), N[i,:],c=color, label= f'$N_0$={N[i,0]:.0f}')
    plt.plot(range(time_max), N_linear[i,:],'--', c=color)

plt.legend(loc='upper right')
plt.xlabel(r"Time $\tau$")
plt.ylabel("Population N")
plt.show()
```

Task 3

a)

The final 100 steps for a cannibalistic population where the dynamics follow the rule

$$\eta_{\tau+1} = \mathrm{R}\eta_\tau \mathrm{e}^{-\alpha\eta_\tau}, \tag{19}$$

with initial population $\eta_0 = 900$, and incidence rate $\alpha = 0.01$ can be seen in figure 6.

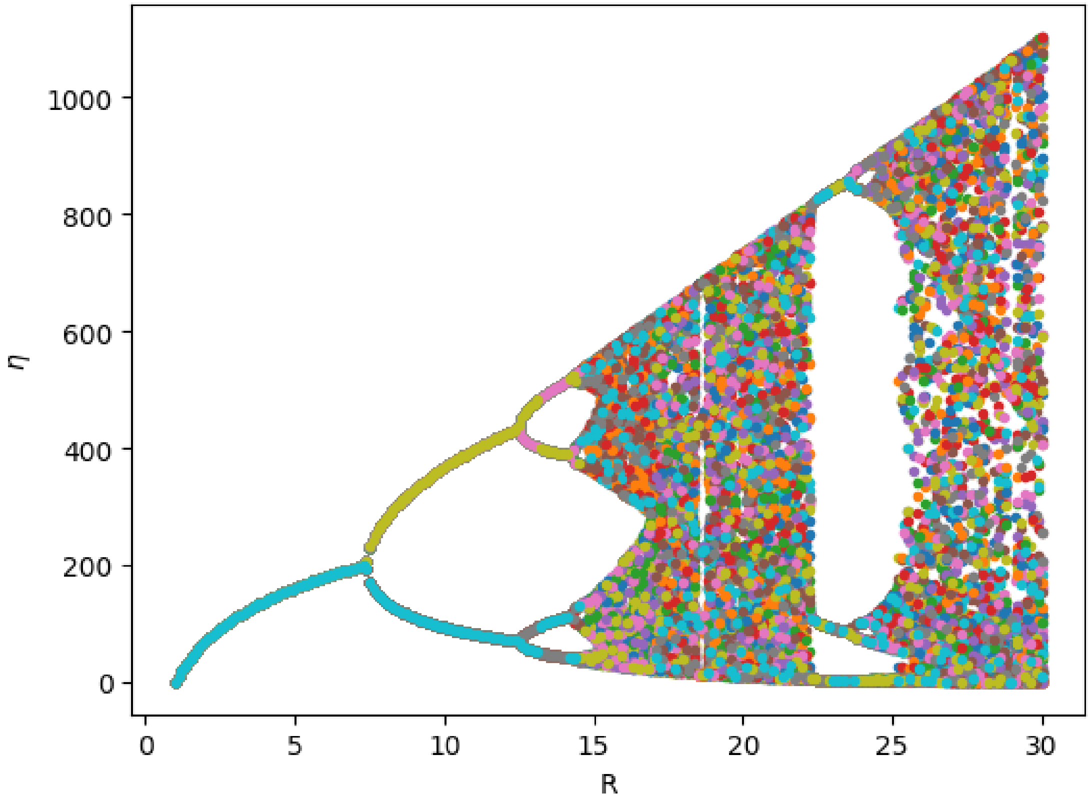
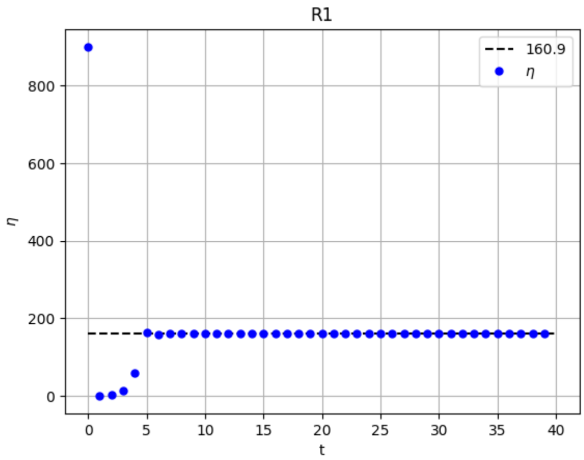


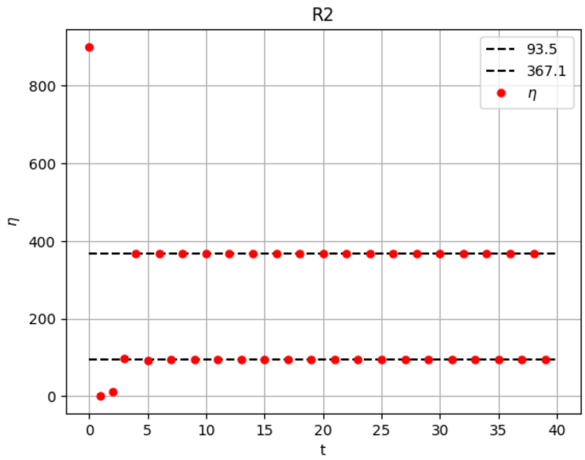
Figure 6: Final 100 steps of the population for values of R from 1 to 30, with step size 0.1. $\alpha = 0.01, \eta_0 = 900$.

b)

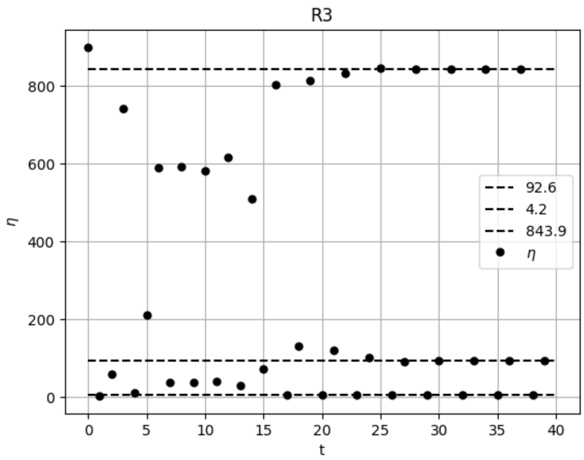
Below are close ups of the values of R giving rise to a stable fixed point, a 2-point cycle. a 3-point cycle, and a 4-point cycle. What happens at the final stages in the different cases is that the population cycles between the different stable states.



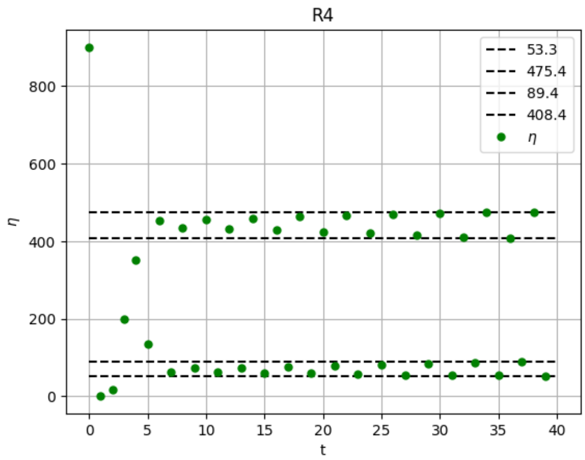
(a) R1 = 5.



(b) R2 = 10.



(c) R3 = 23.

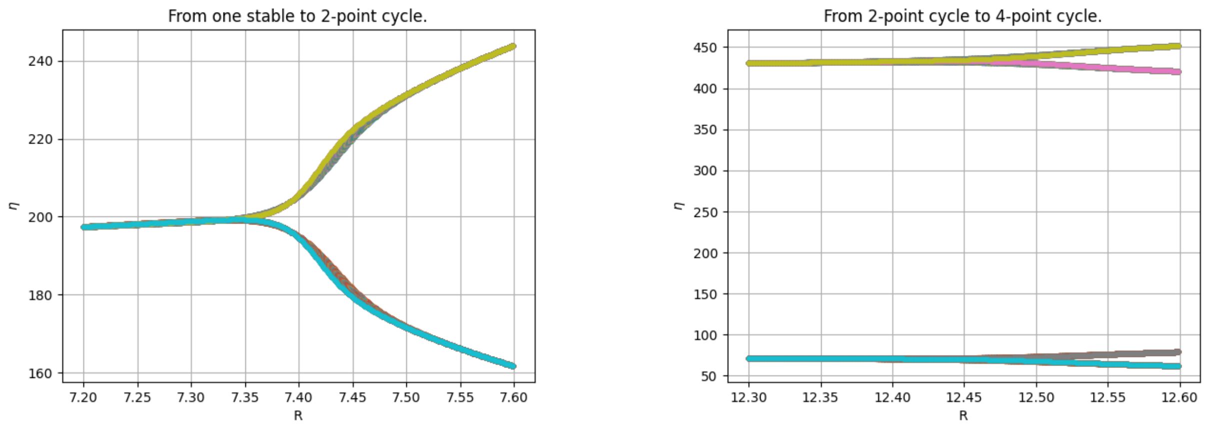


(d) R4 = 13.

Figure 7: Population plotted over time for 40 time steps. The different cases of stable points, and horizontal lines at the final values of the different branches are shown in each subfigure.

c)

The population dynamics bifurcate from a stable equilibrium to a stable 2-point cycle at around R = 7.4, and the dynamics bifurcate to a stable 4-point cycle at around R = 12.5.



(a) Stable equilibrium to a stable 2-point cycle at around R = 7.4.

(b) 2-point cycle bifurcates to a 4-point cycle, at around 12.5.

Figure 8: Two bifurcations, one stable equilibrium to 2-point cycle, and 2-point cycle to 4-point cycle.

d)

R_∞ occurs at around R = 14.80, depending on what counts as reaching R_∞. At this point, the points seem to diverge more than before.

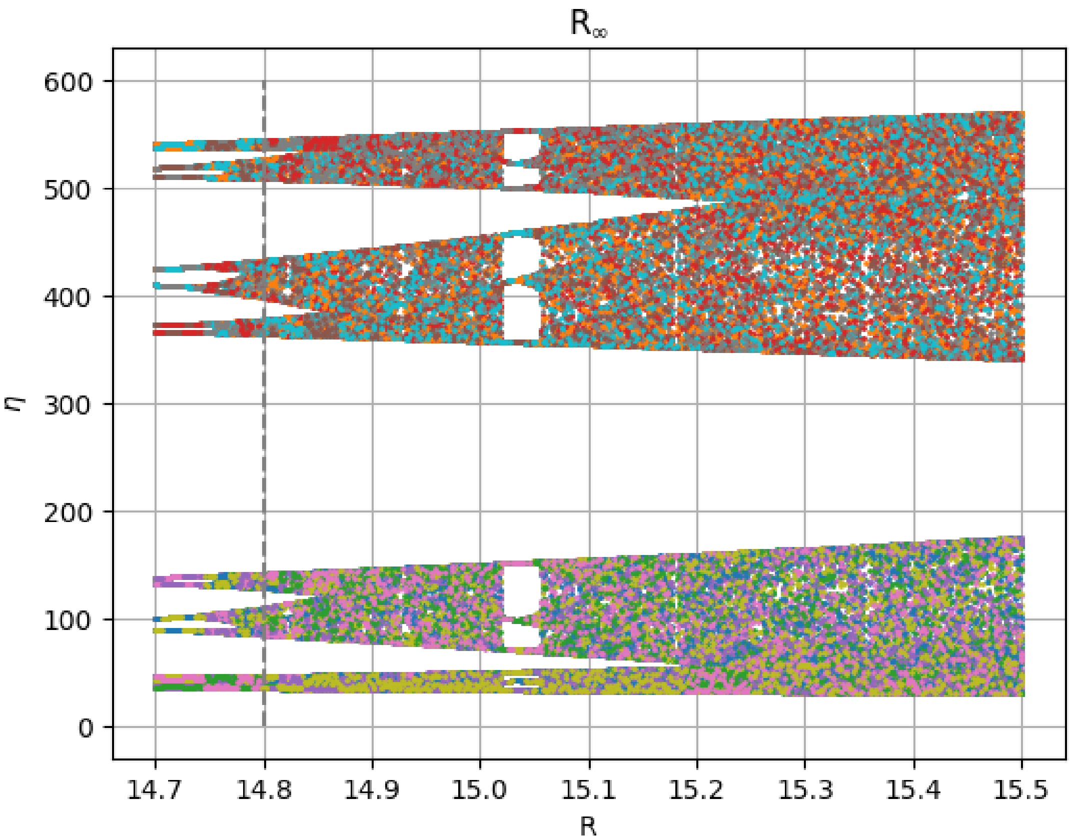


Figure 9: R_∞, a dashed line is shown at the value of R = 14.80.

Python code

```
import numpy as np
import matplotlib.pyplot as plt
def runAndPlot(R, titleString):
    alpha = 0.01
    eta_start = 900
    t_max = 300

    eta = np.zeros((len(R), t_max))
    eta[:,0] = eta_start

    eta_iterated = model(t_max, eta, R, alpha)
    plt.plot(R, eta_iterated[:,-100:], '.', markersize = 2)
    # Plots a vertical line, can be removed if wanted
    #plt.vlines(14.80, 0, 600, 'grey', linestyles='dashed')
    plt.ylabel("$\\eta$")
    plt.xlabel("R")
    plt.grid('minor')
    plt.title(titleString)
    plt.show()

def model(t_max, eta, R, alpha):
    for t in range(t_max-1):
        eta[:, t+1] = np.transpose(R)*eta[:, t]*np.exp(-alpha*eta[:, t])
    return eta
#runAndPlot(R = np.array([5]), titleString='r1')
runAndPlot(R = np.arange(7.2, 7.6, 0.001), titleString= 'r2')
runAndPlot(R = np.arange(22,23, 0.001), titleString='r3')
runAndPlot(R = np.arange(12.3,12.6, 0.001), titleString='r4')
def taskB():
    R_list = np.array([5, 10, 23, 13])

    alpha = 0.01
    eta_start = 900
    t_max = 40

    eta = np.zeros((len(R_list), t_max))
    eta[:,0] = eta_start

    eta_iterated = model(t_max, eta, R_list, alpha)
    colorlist = ['blue', 'red', 'black', 'green']

    for r in range(4):
        plt.figure()
        #plot horizontal lines at endpoints
        for i in range(r+1):
            plt.hlines(eta[r,-1-i], 0,40, 'k', ...
            label=str(np.round(eta[r, -i-1], 1)), linestyles= 'dashed')

        plt.plot(eta[r,:], '.', color = colorlist[r], markersize = 10, label = '$\\eta$')
```

```
        plt.title('R'+str(r+1))
        plt.xlabel('t')
        plt.grid()
        plt.legend()
        plt.ylabel('$\\eta$')
        plt.plot()
taskB()



# Refining for task c and d, can select values and number of points
start_value = 14.7
end_value = 15.5
nPoints = 800
runAndPlot( R = np.linspace(start_value, end_value, nPoints), titleString='R$_\\infty$')
```