Lyapunov spectra

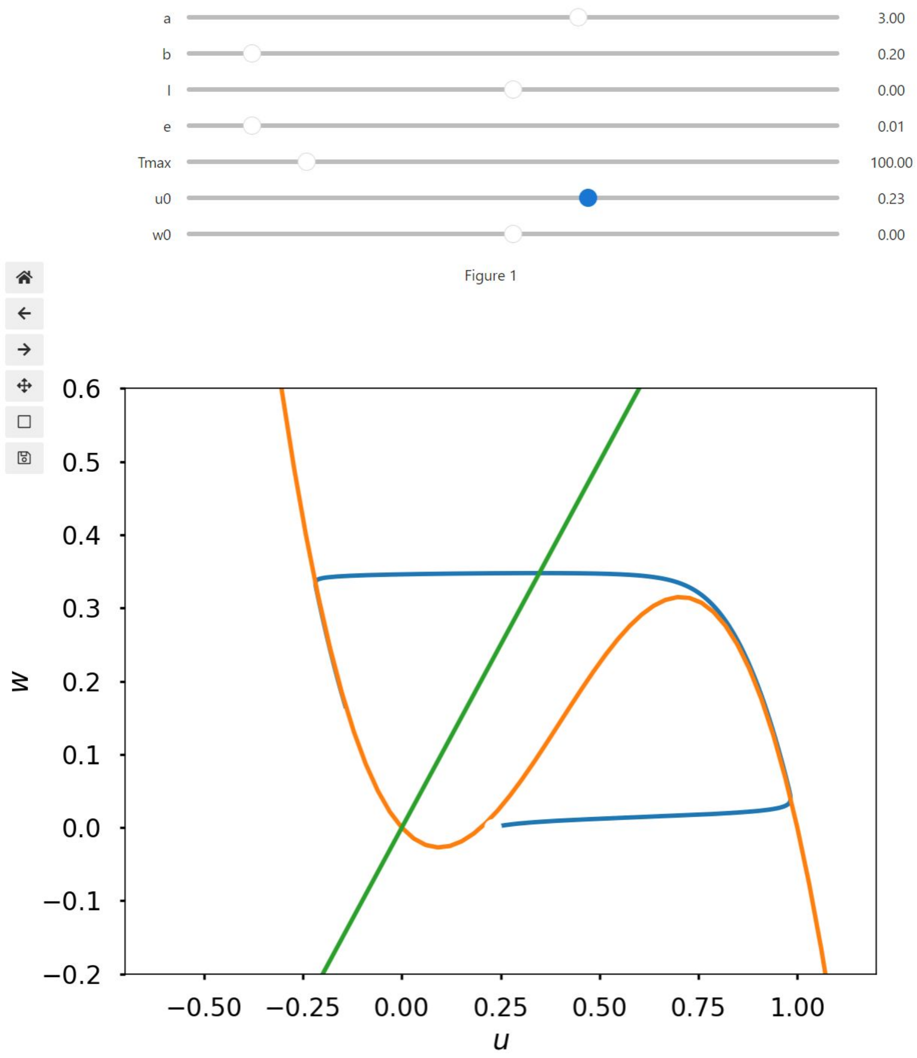
Implementing the 2D model without modules

FritzhughNagumo-Model

```python
def nagumo(x, t, a, b, e, I):
    #x[0] is u          - membrane voltage
    #x[1] is w          - recovery variable
    # dx1dt is u_dot  - change of membrane voltage over time
    # dx2dt is w_dot  - change of recovery

    #t is time
    #a is a
    #b is b            - threshold value
    #e is epsilon
    #I is I             - external injection current

    dx1dt = a*x[0]*(x[0]-b)*(1-x[0])-x[1]+I
    dx2dt = e*(x[0]-x[1])

    return [dx1dt, dx2dt]
```

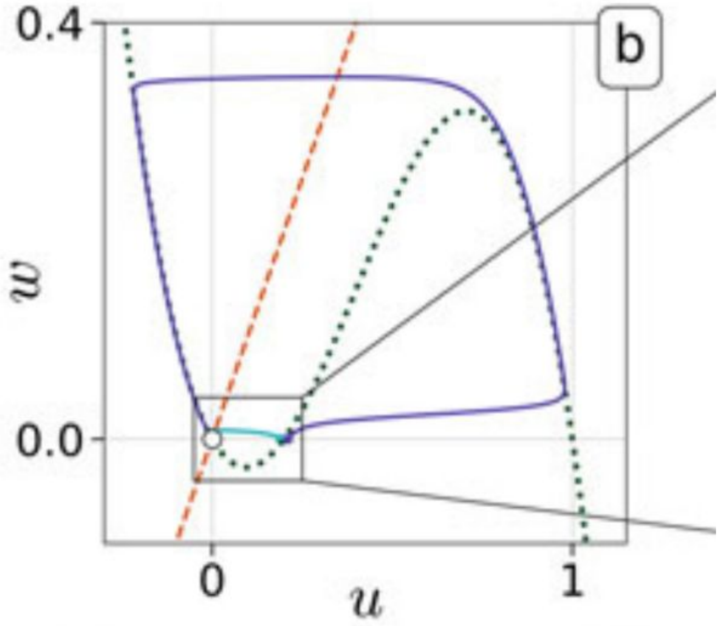
```python
def plot_nullclines(a, b, I, e): # ! FritzhughNagumo-Model spefic !
    # u nullcline
    u = np.linspace(-1, 2, 100)
    w = a*u*(u-b)*(1-u)+I
    plt.plot(u, w)

    # w nullcline
    w = np.linspace(-1, 1)
    u = w
    plt.plot(u, w)

#plot_nullclines (a,b,I,e)
```

$$\dot{u} = au(u-b)(1-u) - w + I$$
$$\dot{w} = \varepsilon(u-w)$$

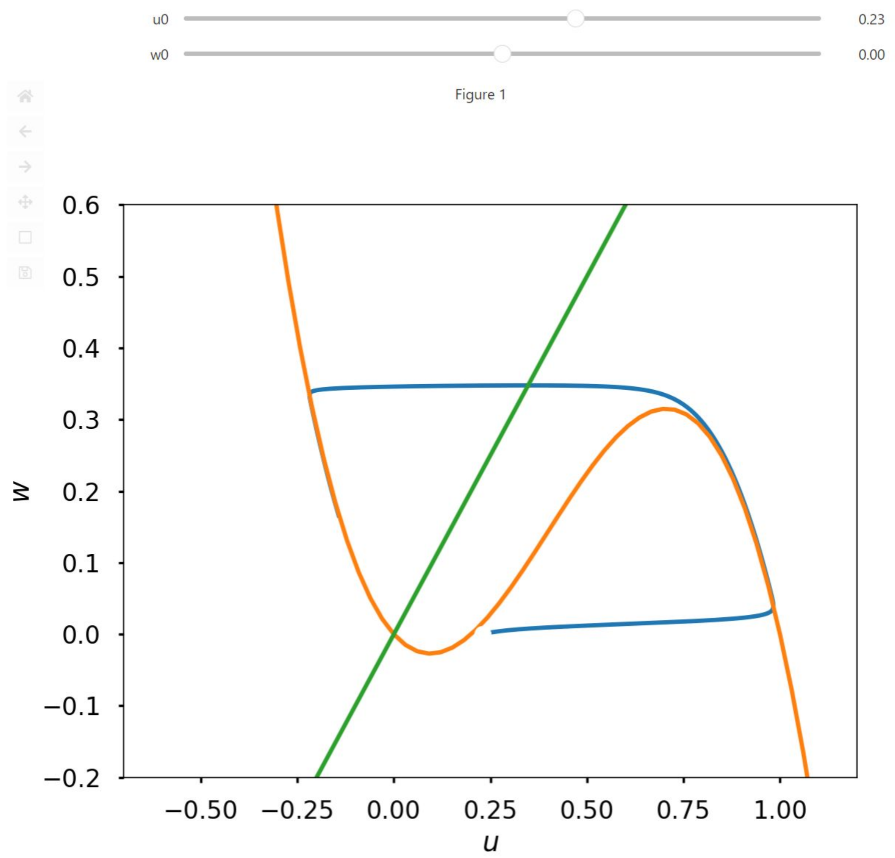


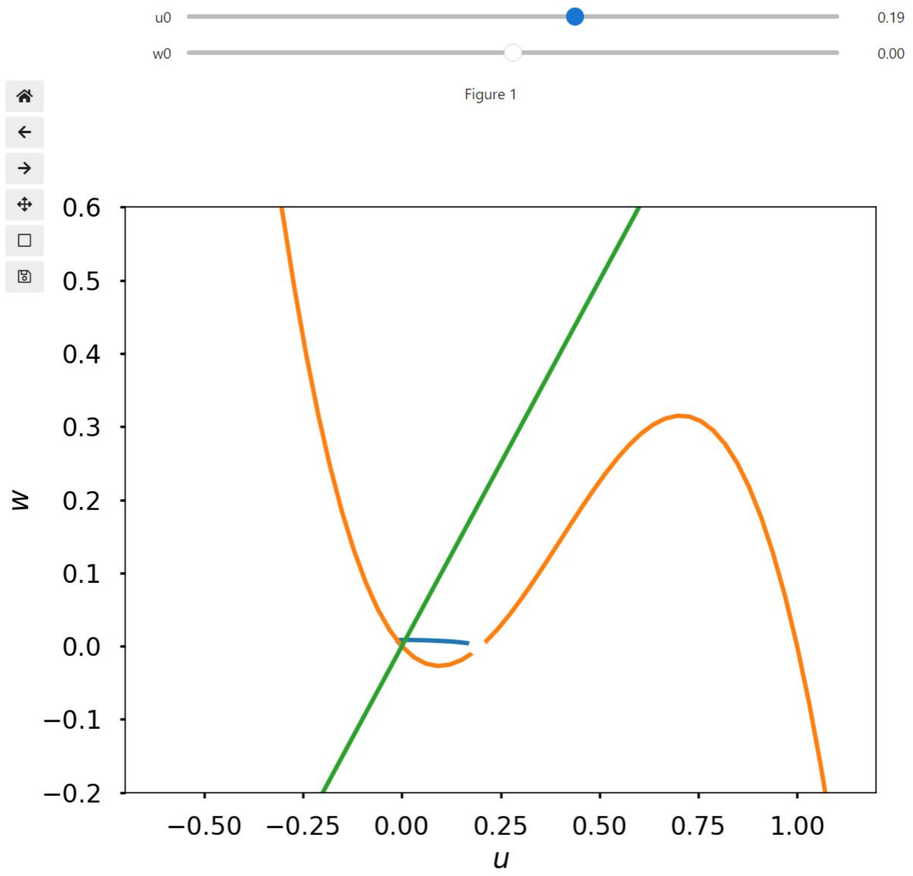
Comparison

Datseris, Parlitz; Nonlinear Dynamics, 2022; Ch.3.1.1. S 39

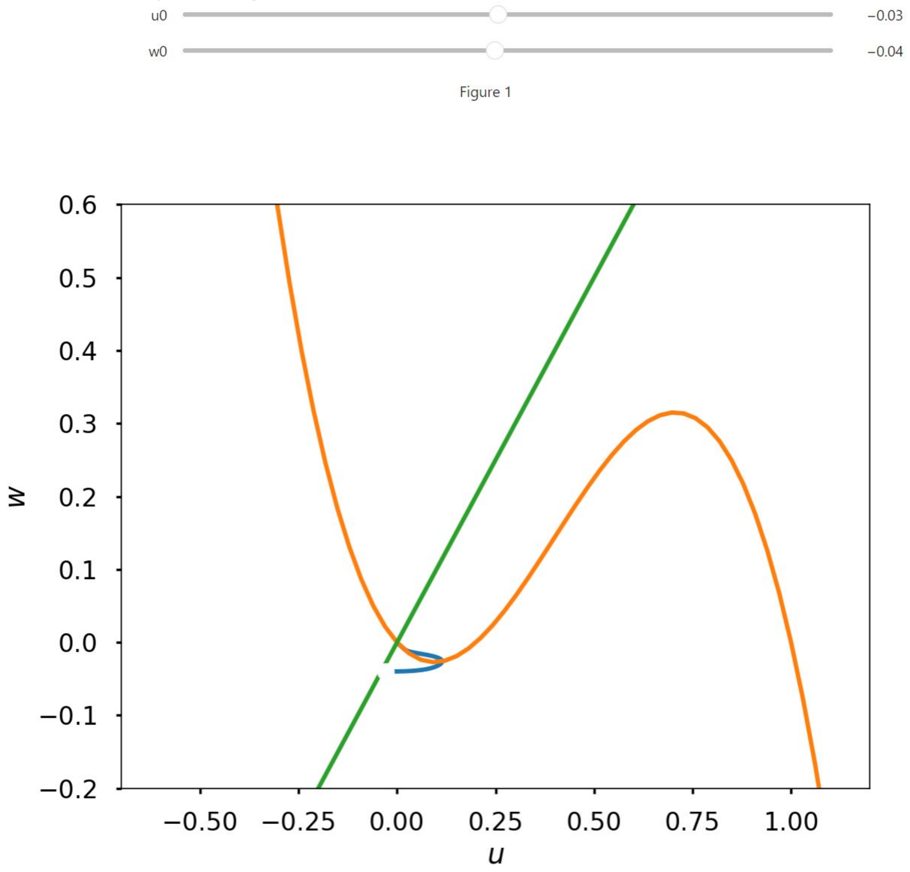


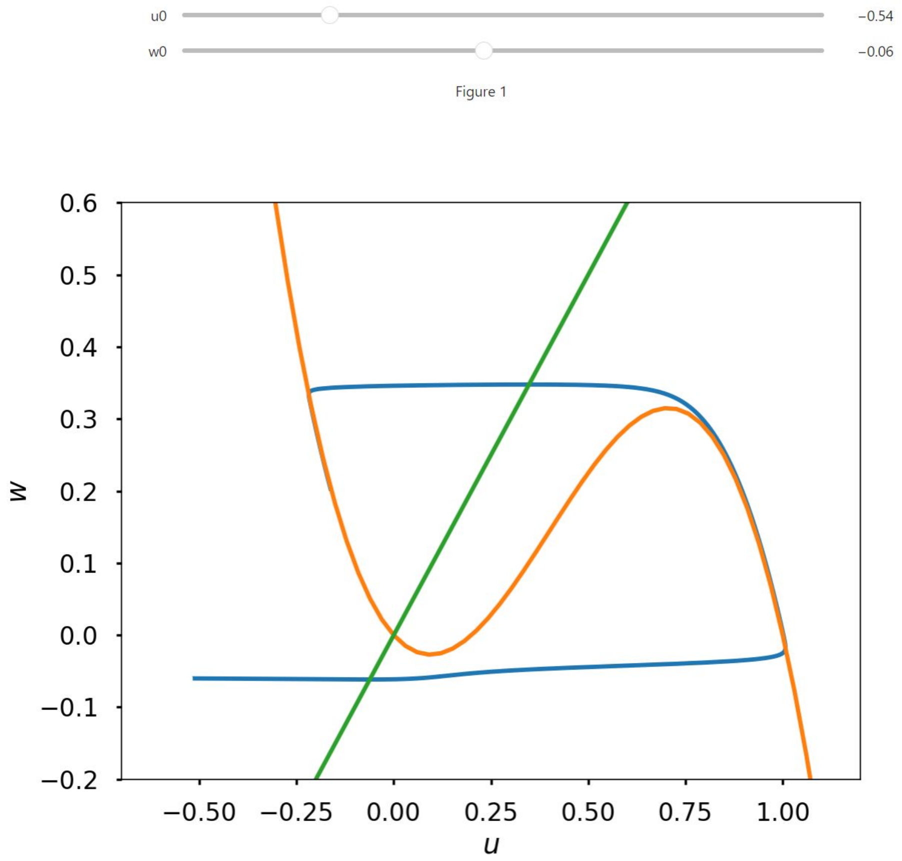
Playing around with sliders



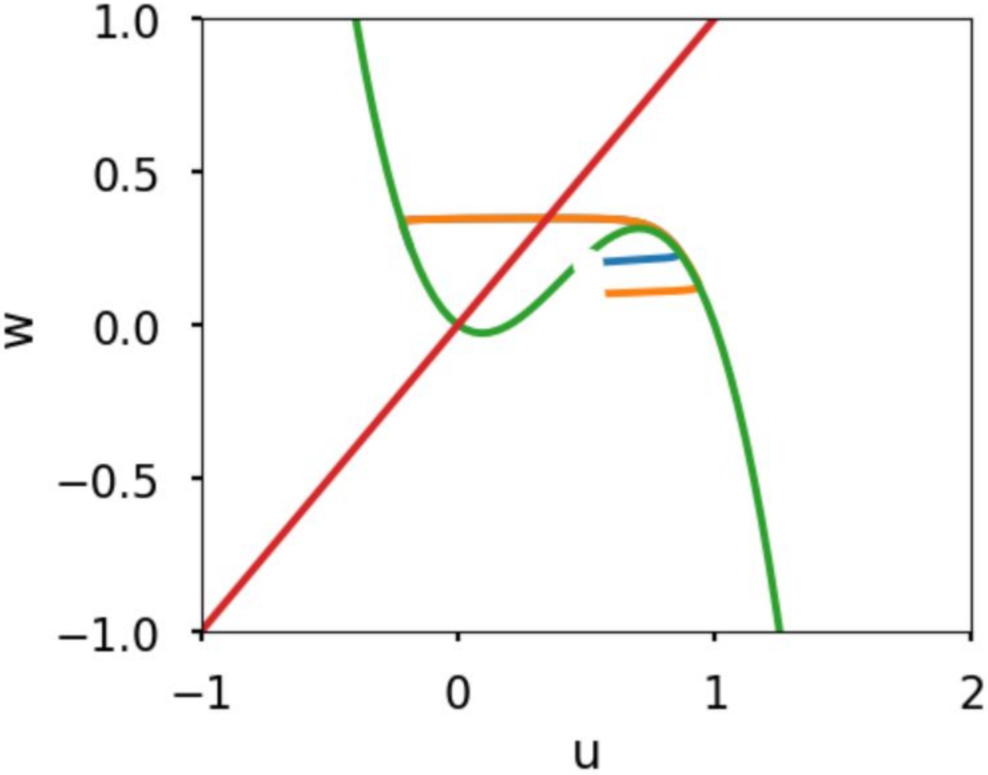


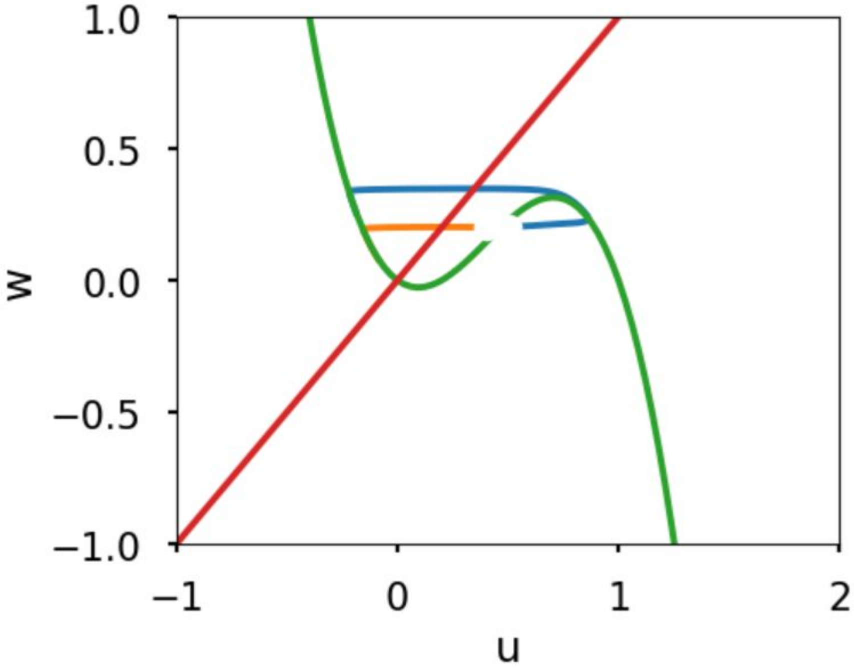
Playing around with sliders





Plotting 2 trajectories (Still with ODEint)

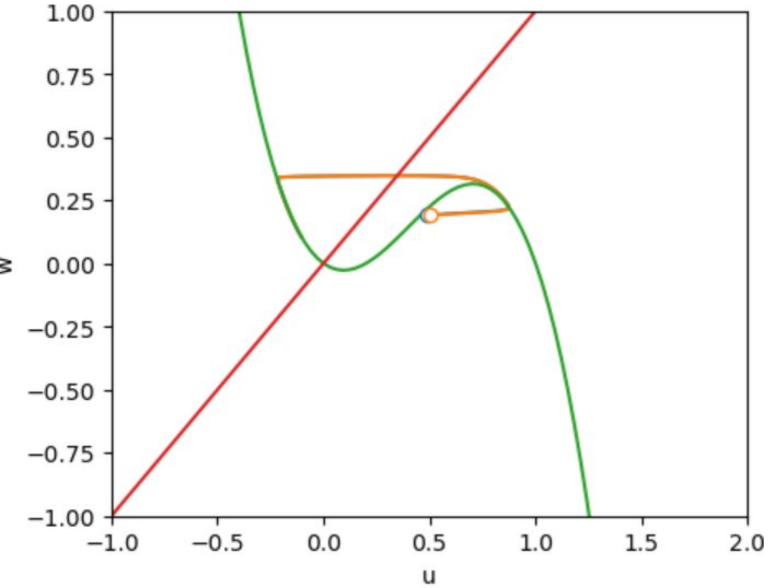


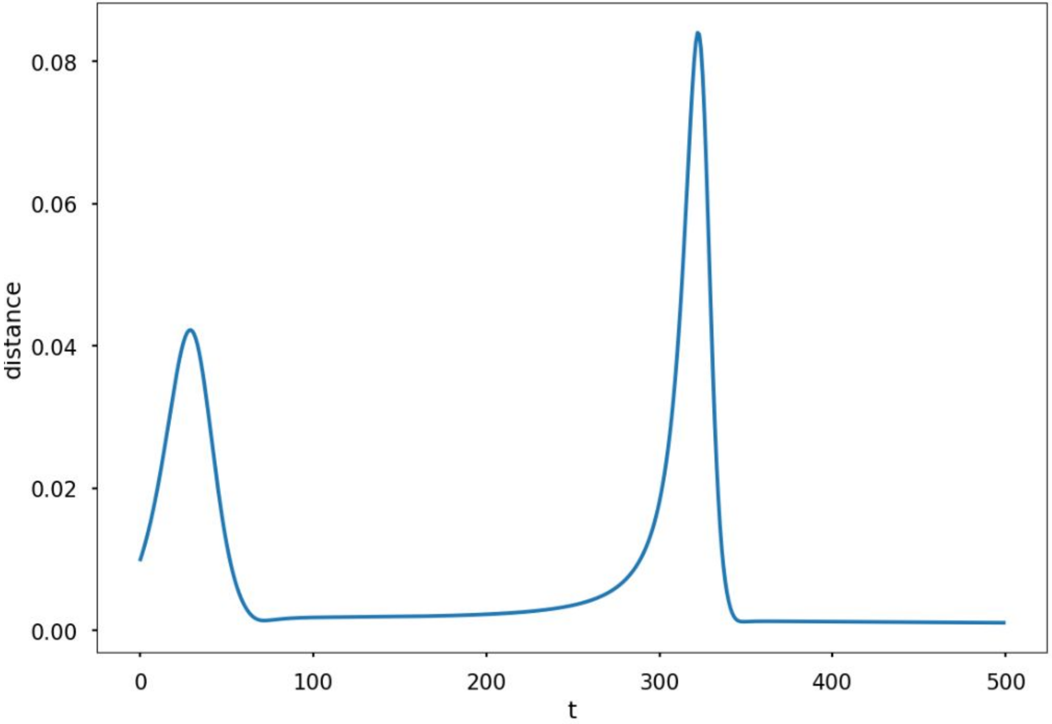


Calculating Distance of 2 slightly perturbed trajectories

```
a=3
b=0.2
I=0
e=0.01
u0=0+0.49
w0=0+0.19
x0 = [0 + u0 ,0 + w0]
```

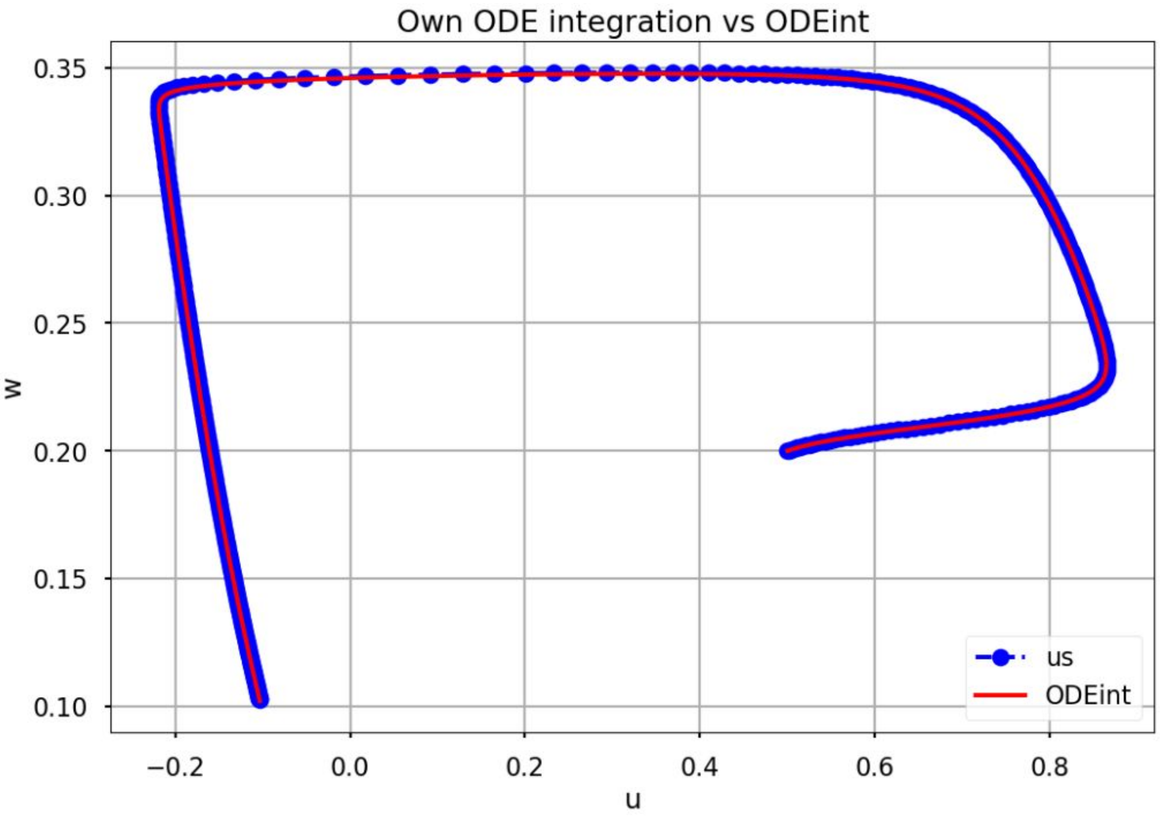
```
u0_1=u0+0.01
w0_1=w0
x0_1 = [0 + u0_1 ,0 + w0_1]
```





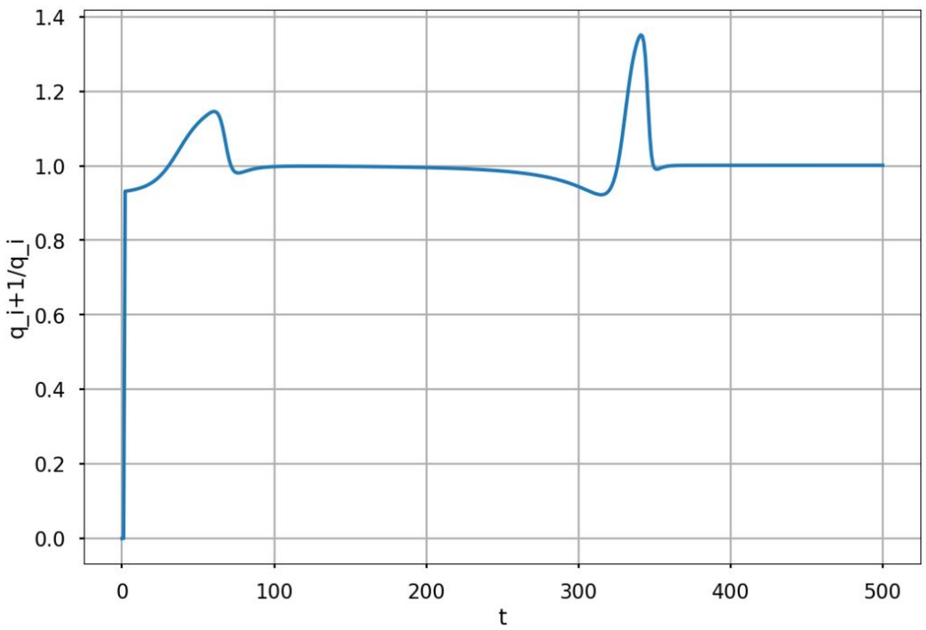
Replacing ODEint with direct Euler-Integration

```
##############################################
# Explicit Euler Method ######################
##############################################

# Define parameters
f= lambda t,x: np.array(nagumo(x,t,a,b,e,I))
h = 0.1 # Step size
t = np.arange(0, 100 + h, h) # Numerical grid
s0 = [u0_1,w0_1] # Initial Condition

s = np.zeros((len(t),2))

s[0] = s0

for i in range(0, len(t) - 1):
    s[i + 1] = s[i] + h*f(t[i], s[i])
```



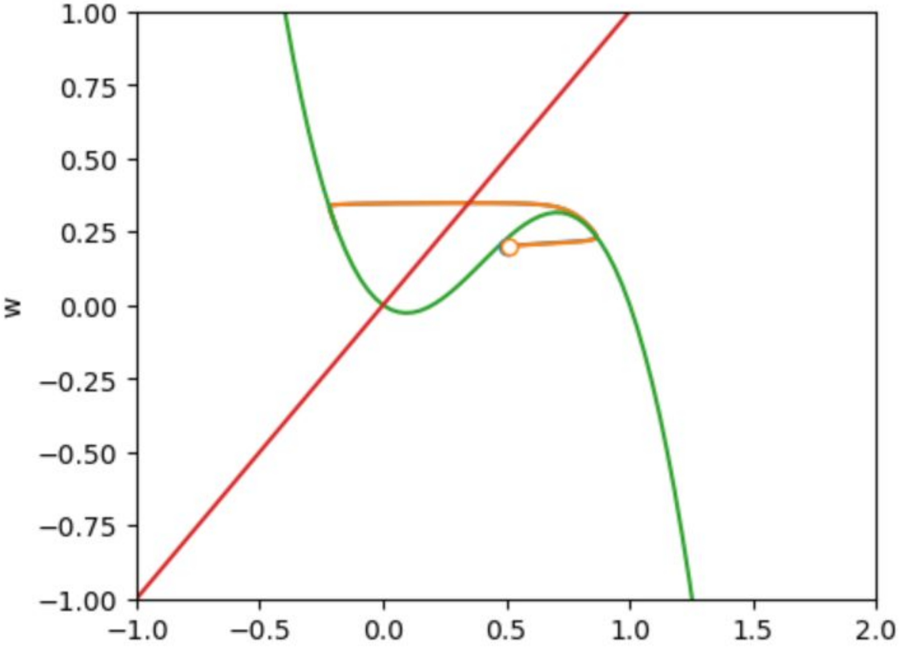
Calculating q_i+1/q_i

```
#function of nagumo via time
f= lambda t,x: np.array(nagumo(x,t,a,b,e,I))
h = 0.1 # Step size of the mathmatical aglorithm

TimeOverAll = 50
t = np.arange(0, TimeOverAll + h, h) # Numerical grid
s0 = [u0,w0] # Initial Condition
s0_1 = [u0_1,w0_1] #Initial Condition of second trajectory

# Explicit Euler Method

s = np.zeros((len(t),2)) #traj of unpur state Array 2 columns, t lines
s_1 = np.zeros((len(t),2)) #traj of purstate
q = np.zeros(len(t))  #Euclidean both traj
qstep = np.zeros(len(t)) #Increasement of last step's Euclidean

s[0] = s0  #initial conditions in first lines of traj-arrays
s_1[0] = s0_1


for i in range(0, len(t) - 1):
    s[i + 1] = s[i] + h*f(t[i], s[i]) #One Euler Step for the first traj
    s_1[i + 1] = s_1[i] + h*f(t[i], s_1[i]) #2nd traj
    q[i+1] = np.linalg.norm(s[i+1] - s_1[i+1]) #calc distance of traj AT THIS STEP
    qstep[i+1] = q[i]/q[i+1] #quotient between q_i+1 and q_i
```



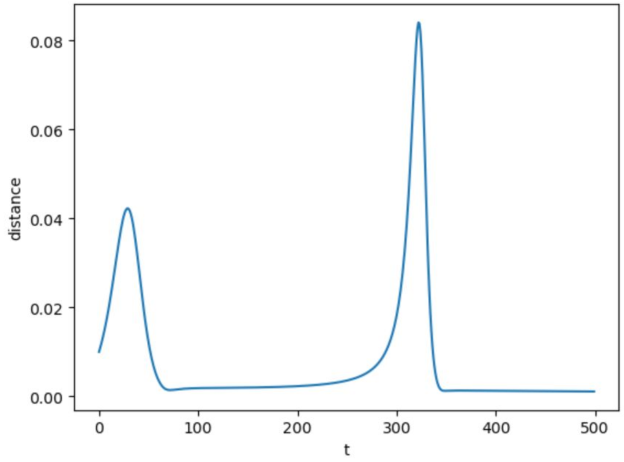
Taking a closer look at the parameters

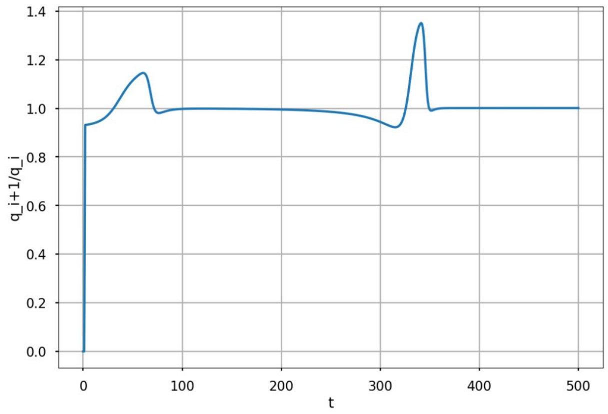
```
#starting values
a=3
b=0.2
I=0
e=0.01

#starting point coordinates
u0=0.5
w0=0.2
x0 = [u0 ,w0] #starting point as vector


#pertubations
u_p = +0.01
w_p = 0

#2nd starting point coordinates
u0_1=u0+u_p
w0_1=w0+w_p
x0_1 = [0 + u0_1 ,0 + w0_1]
```



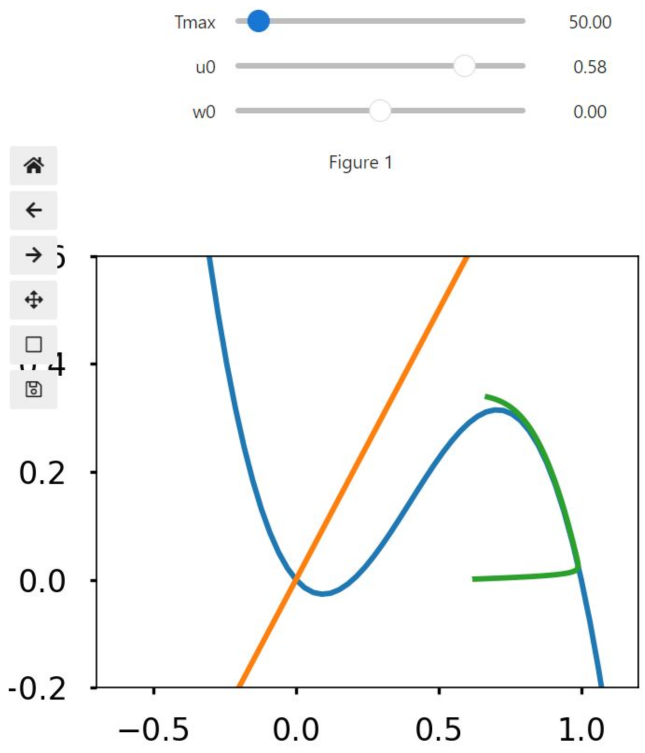
Taking a closer look at the parameters

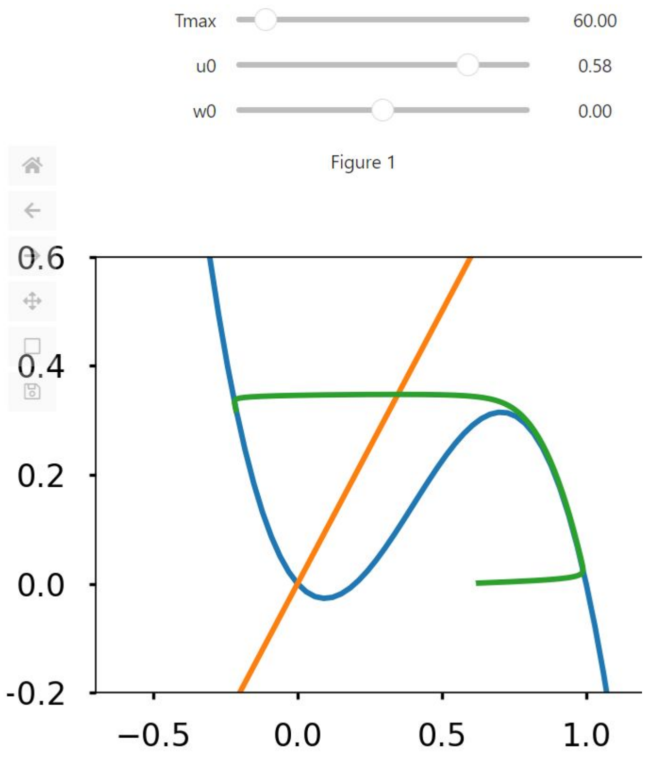
```
#starting values
a=3
b=0.2
I=0
e=0.01

#starting point coordinates
u0=0.5
w0=0.2
x0 = [u0 ,w0] #starting point as vector


#pertubations
u_p = +0.01
w_p = 0

#2nd starting point coordinates
u0_1=u0+u_p
w0_1=w0+w_p
x0_1 = [0 + u0_1 ,0 + w0_1]
```

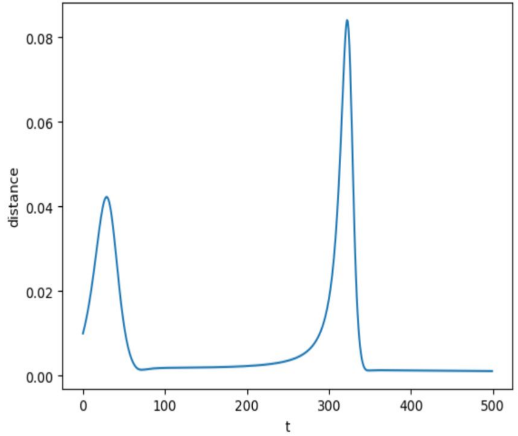


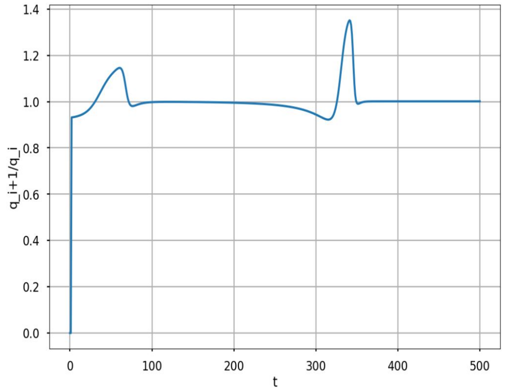


“Hitting” the w-nullcline

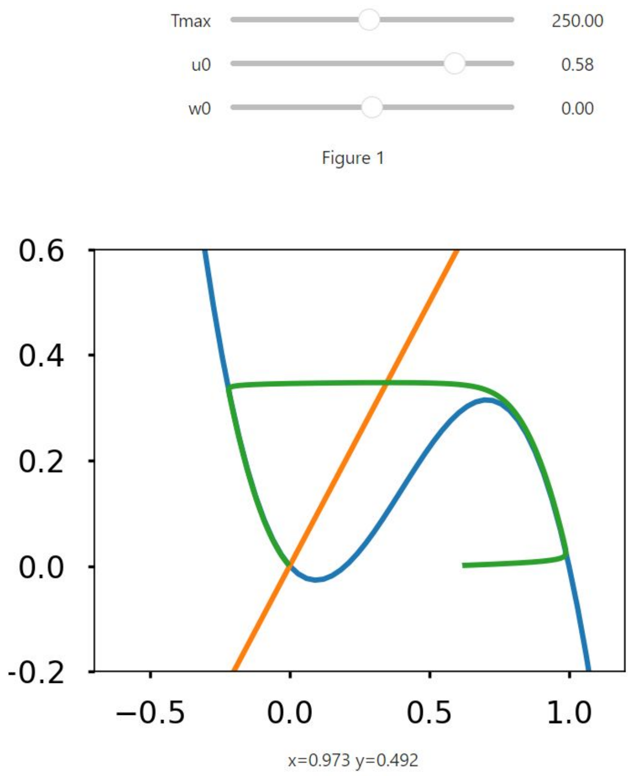


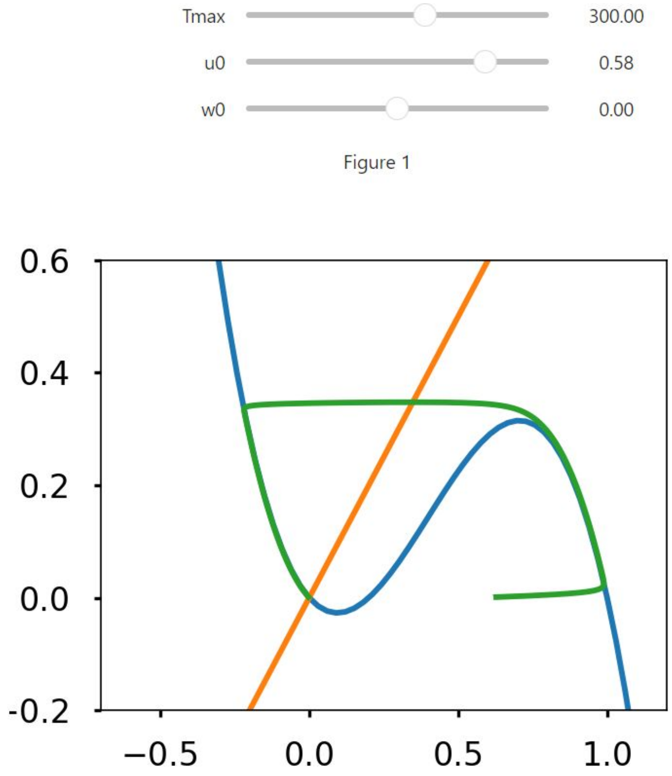


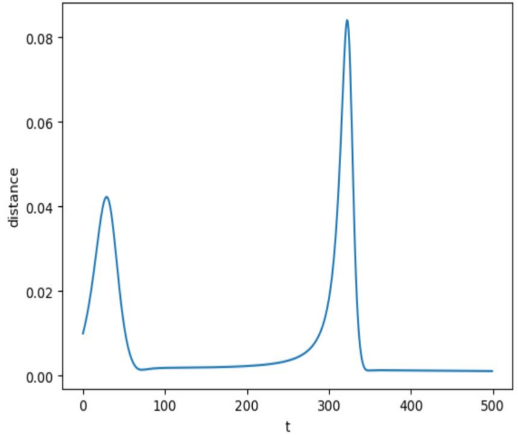


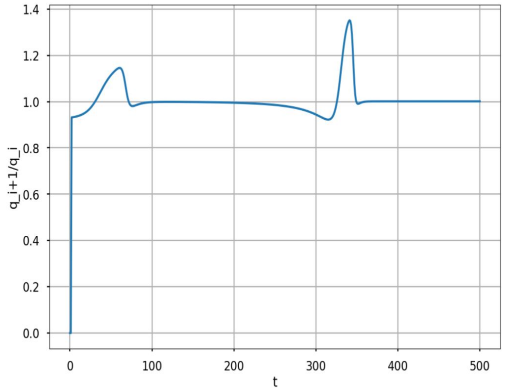


Arrival at the fixed point









Next step

- Lyapunov Exponent
    - via Renormalization

Next Steps

- From 2D System (Fritzhugh-Nagumo) to 3D System (Lorenz)
- QR without renormalization
- QR with renormalization
- (other systems?)
- Other methods
    - Starting with balcerak paper