Python for Data Analysis and Scientific Computing Project:

Kinks and Antikinks of ϕ^4 model

Aslihan Demirkaya

In this project, we present the numerical existence and the behavior of kink-antikink solutions of a very well known partial differential equation (PDE), known as ϕ^4 model.

$$u_{tt} = u_{xx} - V'(u)$$

where

$$V(u) = \frac{1}{2}(1 - u^2)^2$$

Steady State Kinks:

- Solutions that are independent of time variable, i.e. $u(x,t)=\phi(x)$.

- $\phi(x)$ satisfies: $$\phi'' - V'(\phi) = \phi'' - 2\phi^3 + 2\phi = 0.$$

Solving:

$$\phi'' - V'(\phi) = 0$$

$$\phi(-L) = -1$$

$$\phi(L) = 1$$

Differentiation function + fsolve

Initializer

Creating Data for Initial Guess:

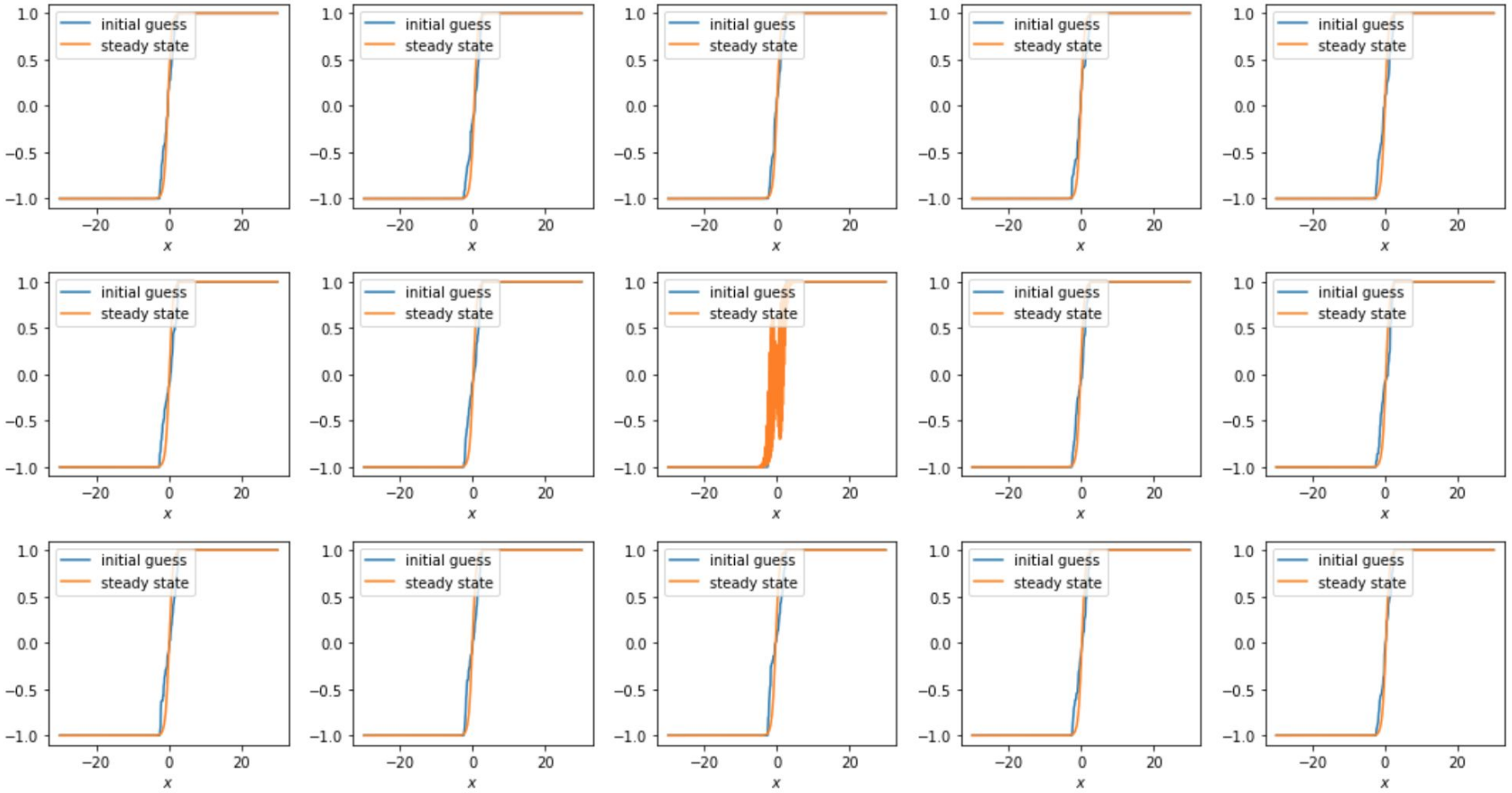
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import randn as rnd
from scipy import io as sio
import scipy.stats as stats
a=rnd(600,100) # we initialize a matrix with 600 rows, 100 columns.
for i in np.arange(100):
    lower, upper = -1, 1
    mu, sigma = 0, 1
    X = stats.truncnorm(
        (lower - mu) / sigma, (upper - mu) / sigma, loc=mu, scale=sigma) # we normalize so th
    a[275:325,i]=X.rvs(50)
    a[0:275,i]=-1*np.ones(275) #The left tail is -1
    a[325:600,i]=np.ones(275)   #The right tail is 1
    a[:,i]=np.sort(a[:,i]) # We have to sort to look like a sigmoid function
sio.savemat('/Users/demirkaya/Desktop/DataScience/Project/InitialData.mat',{'a':a})
```

Loading Initial Data

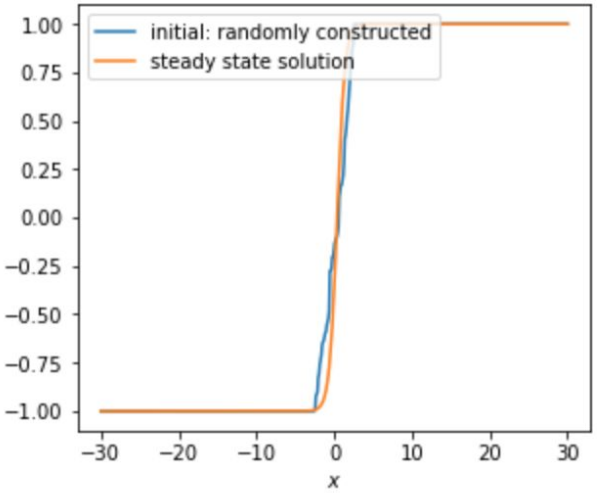
```
initializer=sio.loadmat('/Users/demirkaya/Desktop/DataScience/Project/InitialData.mat',struct_as_record=True)
initial_guess_1=initializer['a']
#initial_guess_1[:,i] where i varies from 0 to 99.
```

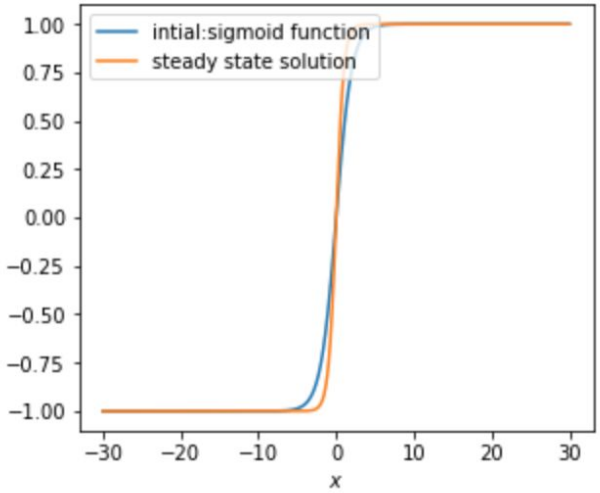
Using fsolve:

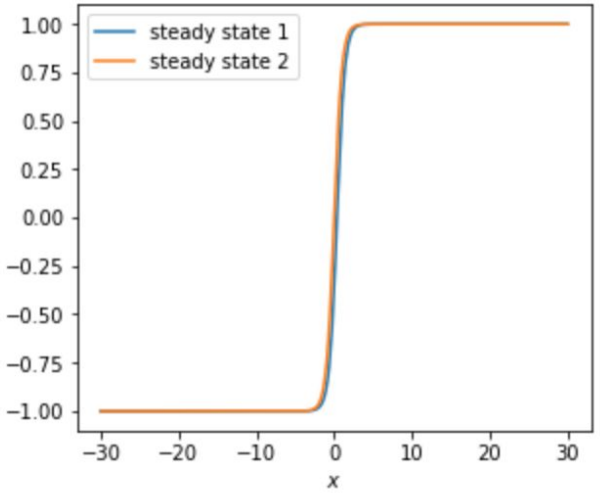
```
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
x=np.linspace(-30.0, 30.0, num=600)
def func2(u):
    out = np.gradient(np.gradient(u,x),x)-2*u**3+2*u
    return(out)

fig = plt.figure(figsize=(15,8))
for i in np.arange(15): #we can try all of them, i.e. 99 instead of 25.
    plt.subplot(3,5,i+1)
    initial_guess=initial_guess_1[:,i]
    steady_state = fsolve(func2,initial_guess)

    plt.plot(x,initial_guess,label='initial guess')
    plt.plot(x,steady_state,label='steady state')
    plt.legend(loc='upper left')
    plt.xlabel('$x$')
fig.tight_layout()
```


initial guess
steady state
x
1.0
0.5
0.0
−0.5
−1.0
−20
0
20

```
from scipy.optimize import fsolve
import matplotlib.pyplot as plt
x=np.linspace(-30.0, 30.0, num=600)
def func2(u):
    out = np.gradient(np.gradient(u,x),x)-2*u**3+2*u
    return(out)

initial_guess=initial_guess_1[:,1]  # we can change 1 to other values (0 to 99) if it does not converge.
initial_guess_sigmoid=(2/(1+np.exp(-x)))-1 #this is a good initializer close to what we want to see.
steady_state_1 = fsolve(func2,initial_guess)
steady_state_2 = fsolve(func2,initial_guess_sigmoid)


fig = plt.figure(figsize=(16,4))
plt.subplot(1,3,1)
plt.plot(x,initial_guess,label='initial: randomly constructed')
plt.plot(x,steady_state_1,label='steady state solution')
plt.legend(loc='upper left')
plt.xlabel('$x$')

plt.subplot(1,3,2)
plt.plot(x,initial_guess_sigmoid,label='intial:sigmoid function')
plt.plot(x,steady_state_2,label='steady state solution')
plt.legend(loc='upper left')
plt.xlabel('$x$')

plt.subplot(1,3,3) #we want to see if the steadty state solutions differ.
plt.plot(x,steady_state_1,label='steady state 1')
plt.plot(x,steady_state_2,label='steady state 2')
plt.legend(loc='upper left')
plt.xlabel('$x$')
```


initial: randomly constructed
steady state solution
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−30
−20
−10
0
10
20
30
x


intial:sigmoid function
steady state solution
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−30
−20
−10
0
10
20
30
x


steady state 1
steady state 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−30
−20
−10
0
10
20
30
x

Moving kink solutions:

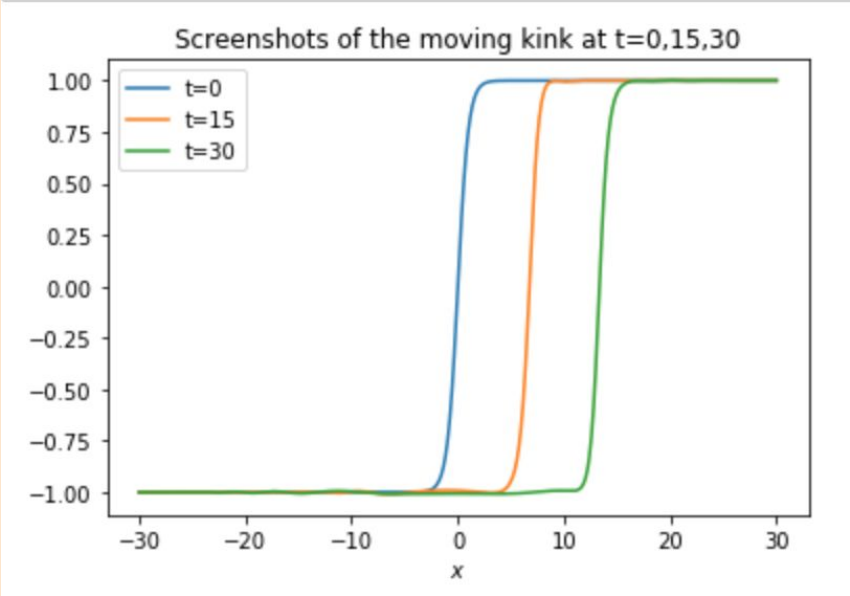
We study the moving single kink in the form: $\phi(x-ct)$

$$u_{tt} = u_{xx} + 2u - 2u^3$$

$$u(x,0) = \phi(x)$$

$$u_t(x,0) = -c\phi'(x)$$

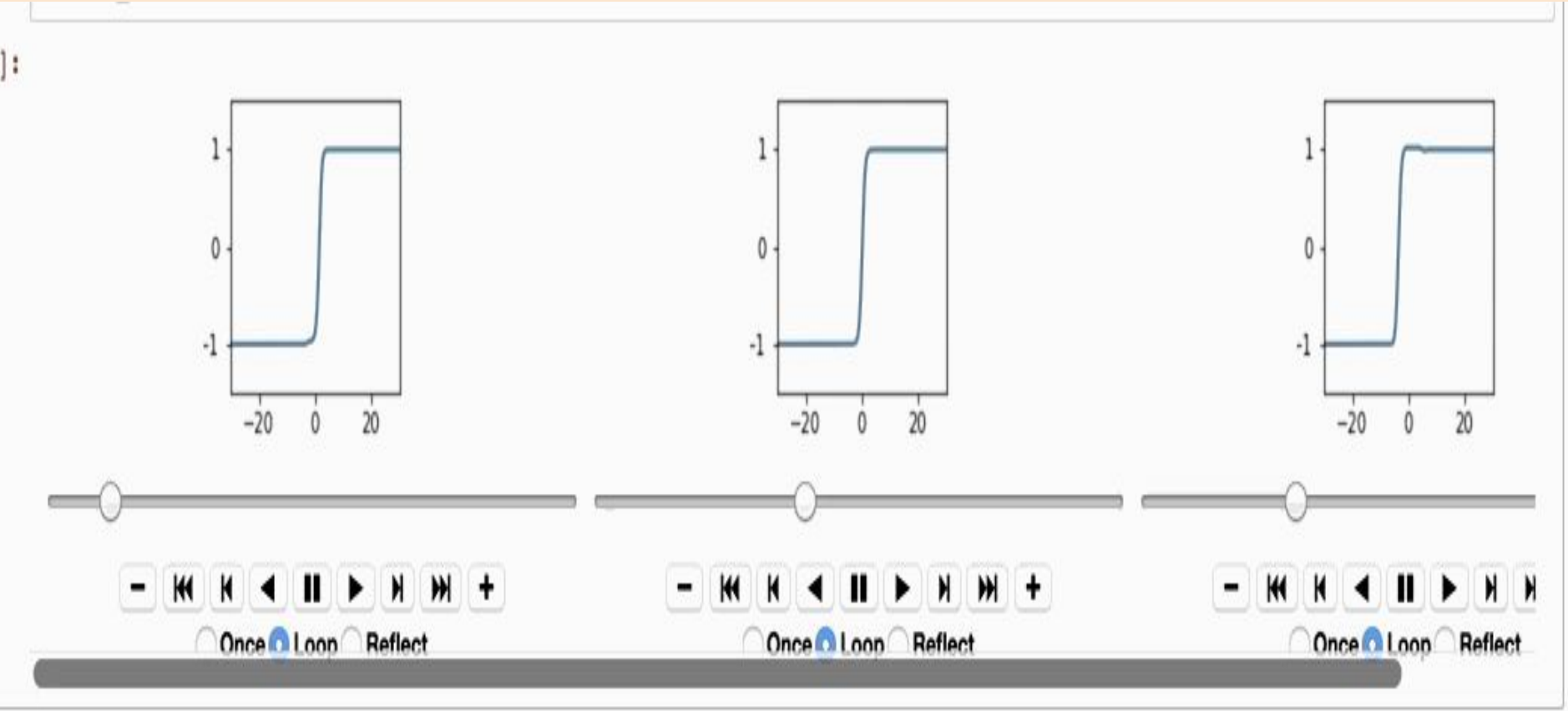
```
from scipy.integrate import odeint
N=len(x)
c=0.5 #we can change the speed here. We take c=0.5, 0 and c=-0,5.
def dU_dt(U, t):
    # Here U is a vector such that u=U[0:N] and u_t=U[N:2*N].
    return np.append(np.array(U[N:2*N]), np.gradient(np.gradient(U[0:N],x),x)+2*U[0:N]-2*(U[0:N]**3))
U0 = np.append(steady_state_2,-c*np.gradient(steady_state_2,x))
ts = np.linspace(0, 30, 100) #time interval is [0,30] partitioned into 100.
Uc1 = odeint(dU_dt, U0, ts)
ys = Uc1
```



```
%matplotlib inline
from matplotlib import animation, rc
from IPython.display import HTML
```

```
N=600
plt.figure(figsize=(2, 2))
fig = plt.gcf()
ax = plt.gca()
# fig, ax = plt.subplots()

ax.set_xlim(( -30, 30))
ax.set_ylim((-1.5, 1.5))

line, = ax.plot([], [], lw=2)

def init():
    line.set_data([], [])
    return (line,)
def animate1(i):
    x = np.linspace(-30.0, 30.0, num=600
    y = kink[0.5][i,:N]
    line.set_data(x, y)
    return (line,)
def animate2(i):
    x = np.linspace(-30.0, 30.0, num=600
    y = kink[0][i,:N]
    line.set_data(x, y)
    return (line,)
```

```
def animate3(i):
    x = np.linspace(-30.0, 30.0, num=600)
    y = kink[-0.5][i,:N]
    line.set_data(x, y)
    return (line,)

anim1 = animation.FuncAnimation(fig, animate1, init_func=init, interval=100, blit=True)
h_anim1 = anim1.to_jshtml()
anim2 = animation.FuncAnimation(fig, animate2, init_func=init, interval=100, blit=True)
h_anim2 = anim2.to_jshtml()
anim3 = animation.FuncAnimation(fig, animate3, init_func=init, interval=100, blit=True)
h_anim3 = anim3.to_jshtml()

plt.close()
#HTML(h_anim2)

H_table = """<table>
  <tr>
    <th>{}</th>
    <th>{}</th>
    <th>{}</th>
  </tr>
  </table>""".format(h_anim1, h_anim2, h_anim3)

HTML(H_table)
```



c=0.5 c=0 c=-0.5

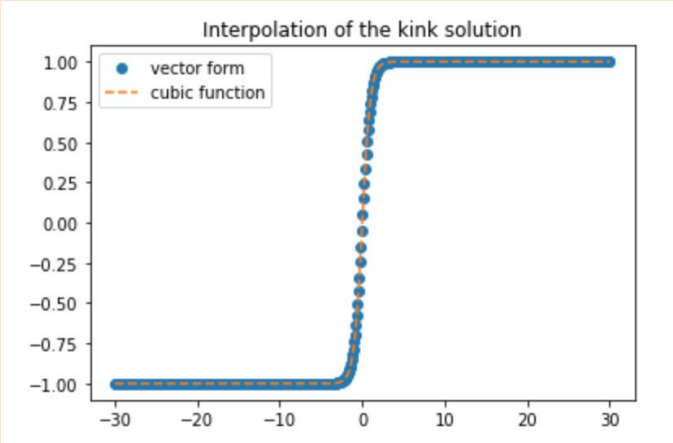
Moving Kink-Antikink Solutions

$$u_{tt} = u_{xx} + 2u - 2u^3$$

$$u(x,0) = \phi(x+10) - \phi(x-10) - 1$$

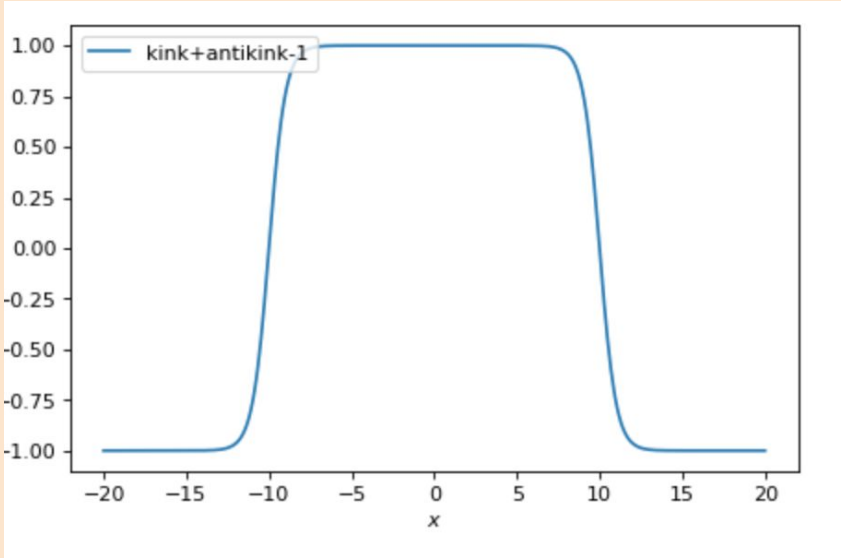
$$u_t(x,0) = -c\phi'(x+10) + c\phi(x-10)$$

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d
x=np.linspace(-30.0, 30.0, num=600)
y=steady_state_2
f = interp1d(x, y, kind='cubic') #by interpolating, we change the vector into a function form

xnew = np.linspace(-30, 30, num=600, endpoint=True)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), '--')
plt.legend(['vector form', 'cubic function'], loc='best')
plt.title('Interpolation of the kink solution')
plt.show()
```



Constructing kink-antikink system

```
x=np.linspace(-20.0, 20.0, num=400)
kink=[]
antikink=[]
for a in x:
    z1=f(a+10)
    kink=np.append(kink,z1)

for a in x:
    z2=-f(a-10)
    antikink=np.append(antikink,z2)
```

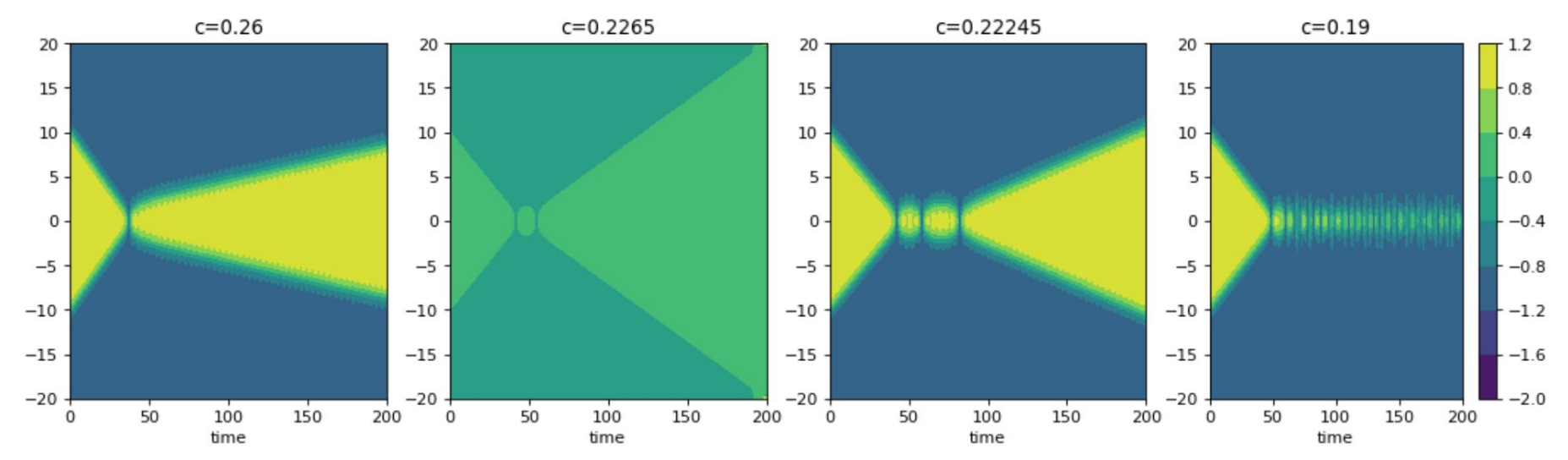


```
def solve(c):
    U0 = np.append(kink+antikink-1,-c*np.gradient(kink,x)+c*np.gradient(antikink,x))
    ts = np.linspace(0, 200, 400) #time interval is [0,100] partitioned into 100.
    U_kak = odeint(dU_dt, U0, ts)
    return U_kak
```

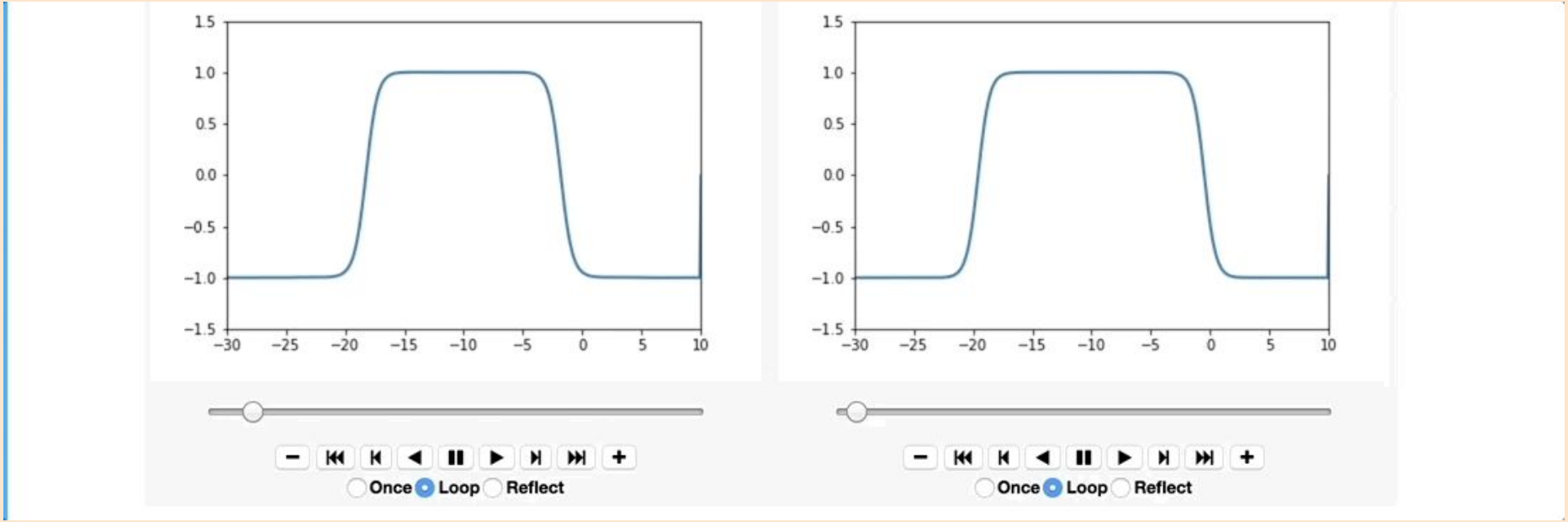
```
from scipy.integrate import odeint
N=len(x)
solutions = dict()
for c in [0.26, 0.2265,0.22245,0.19]:
    print(f"solving for c = {c}")
    solutions[c] = solve(c)
```

```
solving for c = 0.26
solving for c = 0.2265
solving for c = 0.22245
solving for c = 0.19
```

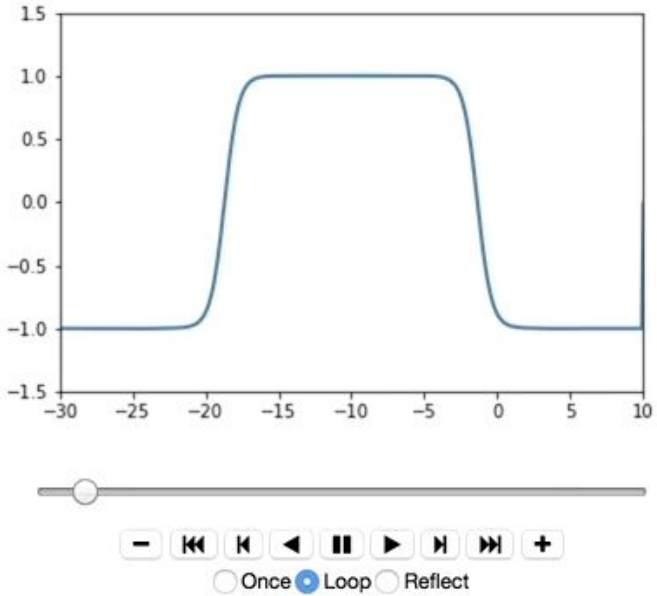
Countermap of the kink-antikink system

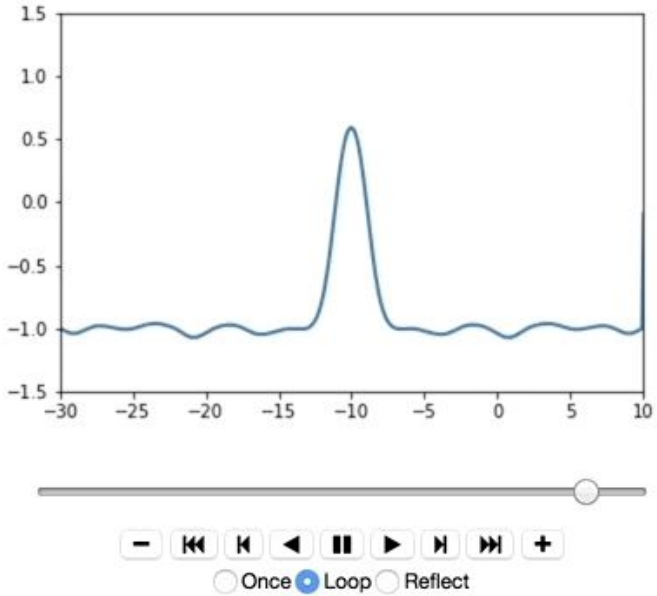


Animations of the kink-antikink system



c=0.26 **c=0.2265**





c=0.22245

c=0.19