```
import numpy as np

def coefficient(i, j):
  c = 0;
  l = abs(i - j)
  if l == 0:
    c = 2
  if l == 1:
    c = -1
  return c

def fillMat(shapeDimentions):
  a = np.zeros(shape=(shapeDimentions,shapeDimentions))
  for idx, x in np.ndenumerate(a):
    for idy, y in np.ndenumerate(x):
      a[idx[0], idx[1]] = coefficient(idx[0], idx[1])

  a[a.shape[0] -1][ a.shape[1] -1] = 1
  return a


M = fillMat(10) * -1



def vectorfield2(w, t, p, M):
    xes = []
    for id, x in enumerate(w):
      if (id % 2) == 0:
        xes.append(x)
    """
    Defines the differential equations for the system of differential equations.

    Arguments:
        w :  vector of the state variables:
                  w = [x1,z1,x2,z2]
        t :  time
        p :  vector of the parameters:
                  p = [m, mu, k, L]
    """

    m, mu, k, L = p
    n = len(w)

    resV = []

                                 (n)):
      # calculate for the last one is kinda tricky:
      if (id % 2) == 0:
        resV.append(w[id+1])
```

```
      else:
        i = (id -1)/ 2 if ((id -1)/ 2 > 0) else 0
        isLast = 0
        if (id == (n-1)):
          isLast = k*L
        a = np.matmul(M[int(i)], xes)
        resV.append( (-mu * w[id] + k * a + isLast)/m)
    f1 = resV
    # Create f = (x1',y1',x2',y2'):
    return f1

# Use ODEINT to solve the differential equations defined by the vector field
from scipy.integrate import odeint

# Parameter values
# Masses:
m = 0.5


# Natural lengths
L = 1


# Spring coefficients
k = 1

W0 = 1

k = 2 * W0 / L**2

# Friction coefficients
mu = 0.25
# Initial conditions
# x1 and x2 are the initial displacements; z1 and z2 are the initial velocities
x1 = 0.0
z1 = 0.0
x2 = 0.2
z2 = 0.0



# ODE solver parameters
abserr = 1.0e-8
relerr = 1.0e-6
stoptime = 20.0
numpoints = 1000

                         output of the ODE solver.
                         only because I want to make
# a plot of the solution that looks nice.
t = [stoptime * float(i) / (numpoints - 1) for i in range(numpoints)]
```

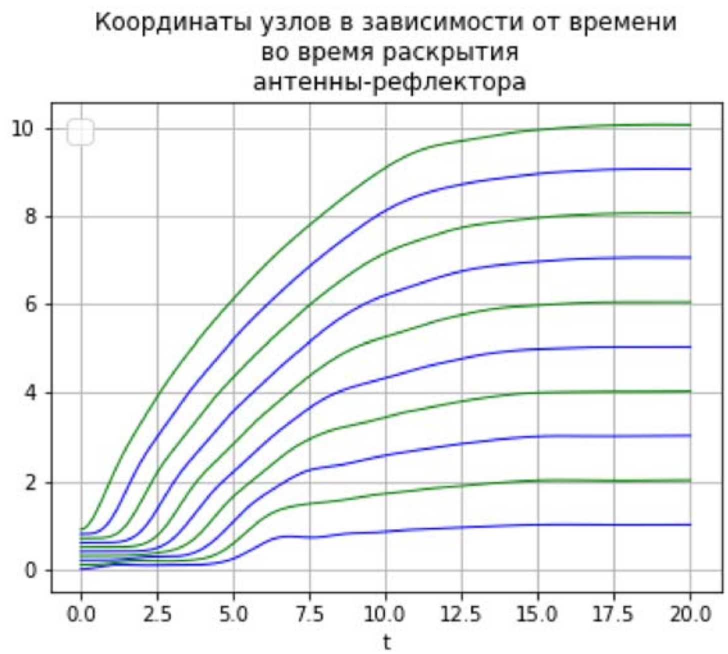
Сохранено

```
p = [m, mu, k, L]

# Pack up the parameters and initial conditions:
# e.g. for 2 elements:
# w0 = [x1, z1, x2, z2]

w0 = []
n = 10
for i in range(n):
  xi = max(w0) + 0.1 if len(w0) > 0 else 0
  w0.append(xi)
  w0.append(0)

# Call the ODE solver.
wsol = odeint(vectorfield2, w0, t, args=(p,M),
              atol=abserr, rtol=relerr)

with open('reflector-dynamics.dat', 'w') as f:
    # Print & save the solution.
    for t1, w1 in zip(t, wsol):
        print(t1, end=" ", file=f)
        for wi in w1:
          print(wi, end=" ",file=f)
        print("",file=f)



from numpy import loadtxt
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties

t, *zres = loadtxt('reflector-dynamics.dat', unpack=True)

xes = []
zes = []


for id, el in enumerate(zres):
  if (id % 2 ) == 0:
    xes.append(el)
  else:
    zes.append(el)

plt.figure(1, figsize=(6, 4.5))

plt.xlabel('t')
plt.grid(True)
```



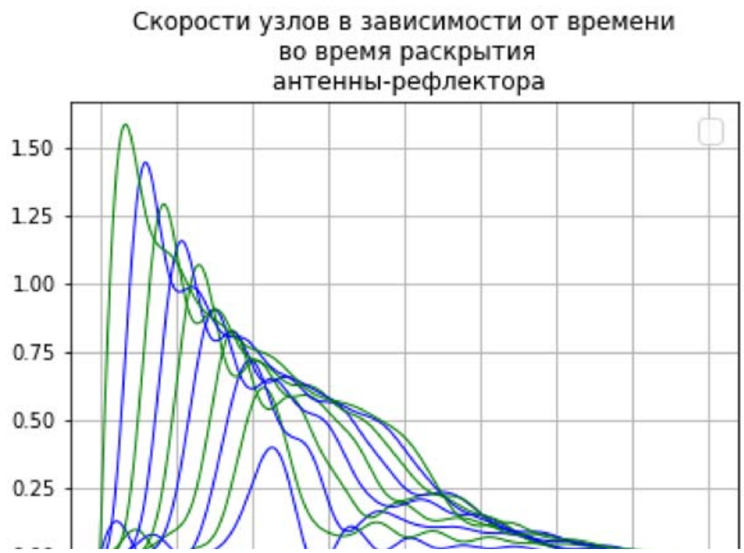
```
for id, x in enumerate(xes):
  color = 'b' if (id % 2) == 0 else 'g'
  plt.plot(t, x, color, linewidth=lw)
```

```
a = []
# a = [r"$x_{0}$".format(10 - i ) for i in range(10)]

plt.legend(a, prop=FontProperties(size=16))
plt.title('Координаты узлов в зависимости от времени\n во время раскрытия\n антенны-р
plt.savefig('reflector\`s_coordinates.png', dpi=100)
```




```
plt.figure(1, figsize=(6, 4.5))

plt.xlabel('t')
plt.grid(True)
lw = 1


for id, z in enumerate(zes):
  color = 'b' if (id % 2) == 0 else 'g'
  plt.plot(t, z, color, linewidth=lw)

# a = [r"$v_{0}$".format(10 - i ) for i in range(10)]


plt.legend(a, prop=FontProperties(size=16))
plt.title('Скорости узлов в зависимости от времени\n во время раскрытия\n антенны-реф
plt.savefig('reflector\`s_velocities.png', dpi=100)
```

Сохранено




```
import matplotlib.animation as animation
# Plot the solution that was generated
!pip install matplotlib


from matplotlib import rc
rc('animation', html='jshtml')
# rc('animation', embed_limit= 2 ** 128 )

# rc('animation', html='html5')
# rcParams['animation.html'] = 'jshtml'
# rcParams["animation.embed_limit"] = 2 * 128
```

```
print(zres[4])
```

```
def middle(left, right):
    c = L / 2
    halfOfLength = (right[0]-left[0])/2 if ((right[0]-left[0])/2 < c) else c #  a
    xMiddle = halfOfLength + left[0]
    yMiddle = (c ** 2 - halfOfLength ** 2 ) ** 0.5
    return [xMiddle, -yMiddle]
```

```
def initialPlot(xes, amount):
```

Сохранено

```
    yNodes = [0]

    middleInitial = middle([0,0], [xes[0][0], 0])
```

```
        xNodes.append(middleInitial[0])
        yNodes.append(middleInitial[1])

        nra = amount

        for i in range(nra):
          middleAction = middle([xes[i][0],0],[xes[i+1][0],0])
          xNodes.append(xes[i][0])
          xNodes.append(middleAction[0])
          yNodes.append(0)
          yNodes.append(middleAction[1])

        xNodes.append(xes[nra][0])
        yNodes.append(0)

        return [xNodes, yNodes]

    def anime_0(xes, amount, idx):
        xNodes = [0]

        yNodes = [0]

        middleInitial = middle([0,0], [xes[0][idx], 0])
        xNodes.append(middleInitial[0])
        yNodes.append(middleInitial[1])

        nra = amount

        for i in range(nra):
          middleAction = middle([xes[i][idx],0],[xes[i+1][idx],0])
          xNodes.append(xes[i][idx])
          xNodes.append(middleAction[0])
          yNodes.append(0)
          yNodes.append(middleAction[1])

        xNodes.append(xes[nra][idx])
        yNodes.append(0)

        return [xNodes, yNodes]

    def anime_2(xes, amount, idx):
        xNodes = [0]

        yNodes = [0]

        middleInitial = middle([0,0], [xes[0][idx], 0])
                                        0])
                                         ])

        nra = amount
```

```
    for i in range(nra):
      middleAction = middle([xes[i][idx],0],[xes[i+1][idx],0])
      xNodes.append(xes[i][idx])
      xNodes.append(middleAction[0])
      yNodes.append(0)
      yNodes.append(middleAction[1])

    xNodes.append(xes[nra][idx])
    yNodes.append(0)

    yNodes = [yx - 1 for yx in yNodes]

    return [xNodes, yNodes]



nodesAmount = 8
# animation.rcParams["animation.embed_limit"] = 2 ** 128

a = initialPlot(xes, nodesAmount)

lowerA = [ix -1 for ix in a[1] ]

b = a

b[1] = lowerA


fig = plt.figure(figsize=(20, 13))
ax = plt.axes(xlim=(-1, 10), ylim=(-5, 6))

innerY = []
for i, el in enumerate(a[1]):
  if (i % 2) == 0:
    innerY.append(a[1][i])
  else:
    innerY.append(b[1][i])

InnerLine = [a[0], innerY]

line, = ax.plot( a[0], a[1] ,'bo-')
line2, = ax.plot( b[0], b[1] ,'bo-')
line3, = ax.plot(InnerLine[0], InnerLine[1], 'bo-')

def middle(left, right):
    c = L / 2
                              [0])/2 if ((right[0]-left[0])/2 < c) else c #  a
                              [0]
    yMiddle = (c ** 2 - halfOfLength ** 2 ) ** 0.5
    return [xMiddle, -yMiddle]

def init():
```

Сохранено

```
def init():
    line.set_data( a[0], a[1])
    line2.set_data(a[0], a[1])
    line3.set_data(InnerLine[0], InnerLine[1])
    return line, line2, line3

# init()

def anime(i):
  return anime_0(xes, nodesAmount, i)

def anime2(i):
  return anime_2(xes, nodesAmount, i)

def animate(i):

    gg = anime(i)
    gg2 = anime2(i)

    innerY = []
    for i, el in enumerate(gg[1]):
      if (i % 2) == 0:
        innerY.append(gg[1][i])
      else:
        innerY.append(gg2[1][i])

    InnerLine = [gg[0], innerY]


    line.set_data(gg[0], gg[1])
    line2.set_data(gg2[0], gg2[1])
    line3.set_data(InnerLine[0], InnerLine[1])

    return line, line2, line3

anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=1000, interval=1, blit=True)
```

Сохранено

0 сек. выполнено в 21:38

Сохранено