ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«ДИНАМИКА СИСТЕМЫ»**

**ПО ДИСЦИПЛИНЕ «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА»**

**ВАРИАНТ ЗАДАНИЯ 20**

Выполнил студент группы М8О-209Б-22

Концебалов О.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Авдюшкин А.Н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

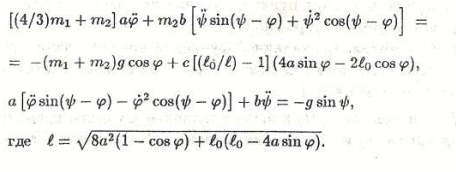
Москва, 2023

**Задание:** проинтегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы с помощью средств Python. Построить графики законов движения системы и указанных в задании реакций для разных случаев системы.

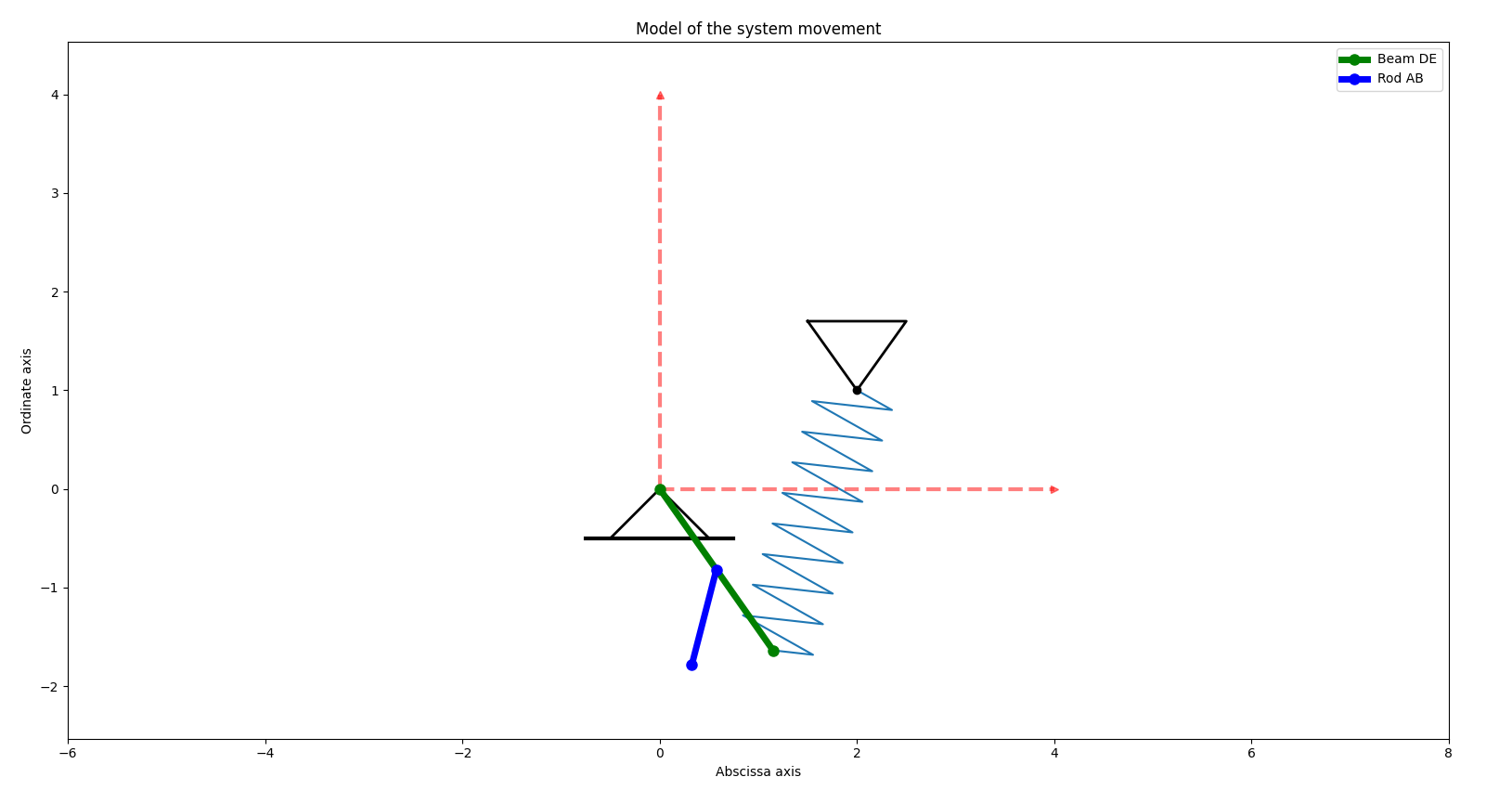
**Задание системы 20 варианта формулируется следующим образом:**

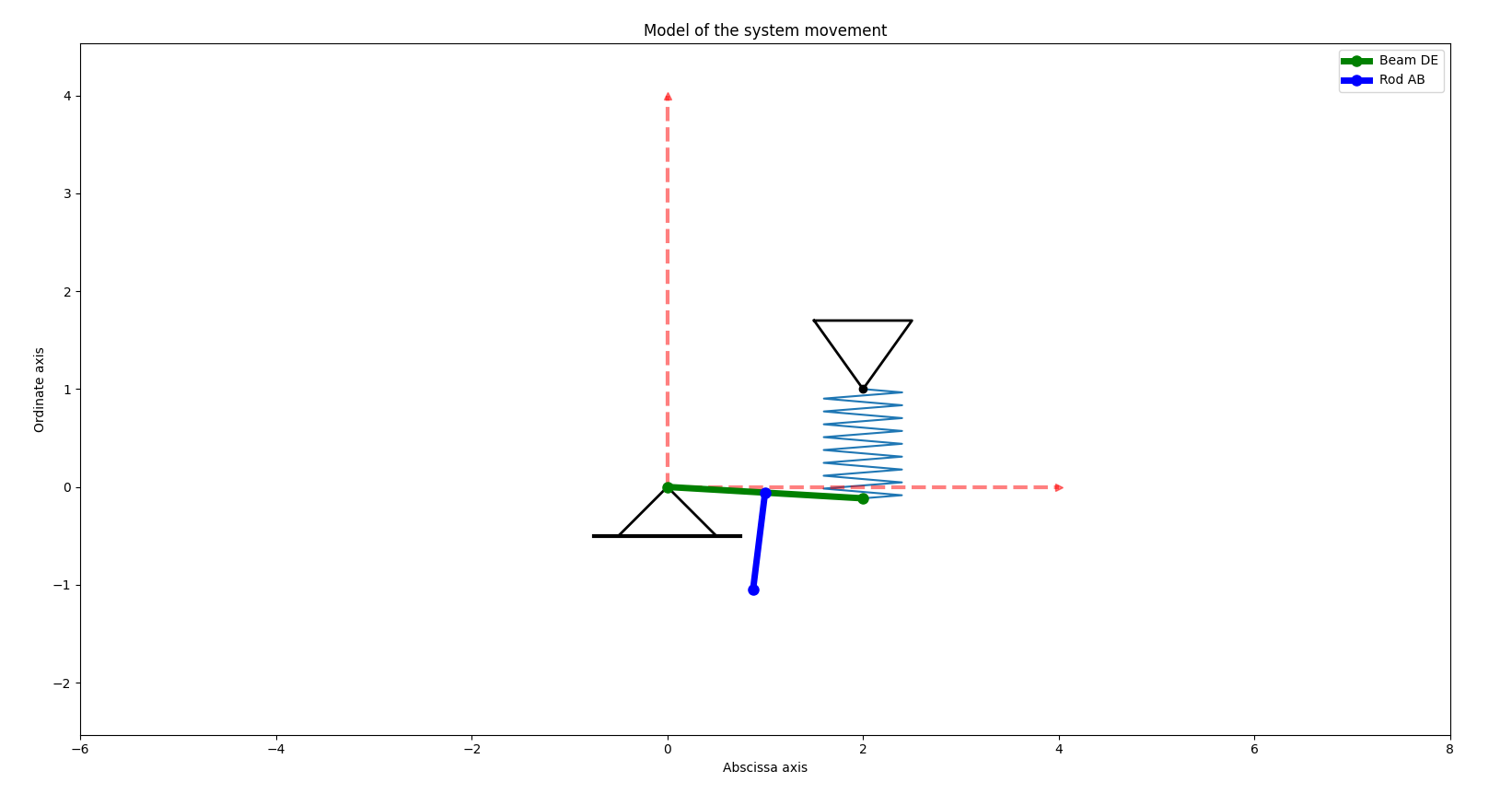
Однородная балка *DE* длины *2a* и массы *m1*закреплена в неподвижном шарнире *D* и концом *E* соединена с пружиной жесткости *c*. К середине балки прикреплен невесомый стержень *AB* длины *b* с точечным грузом массы *m2* на конце *B*. Длина недеформированной пружины равна *l0*; при горизонтальном положении балки *DE* пружина вертикальна.

Дифференциальные уравнения движения системы имеют вид:

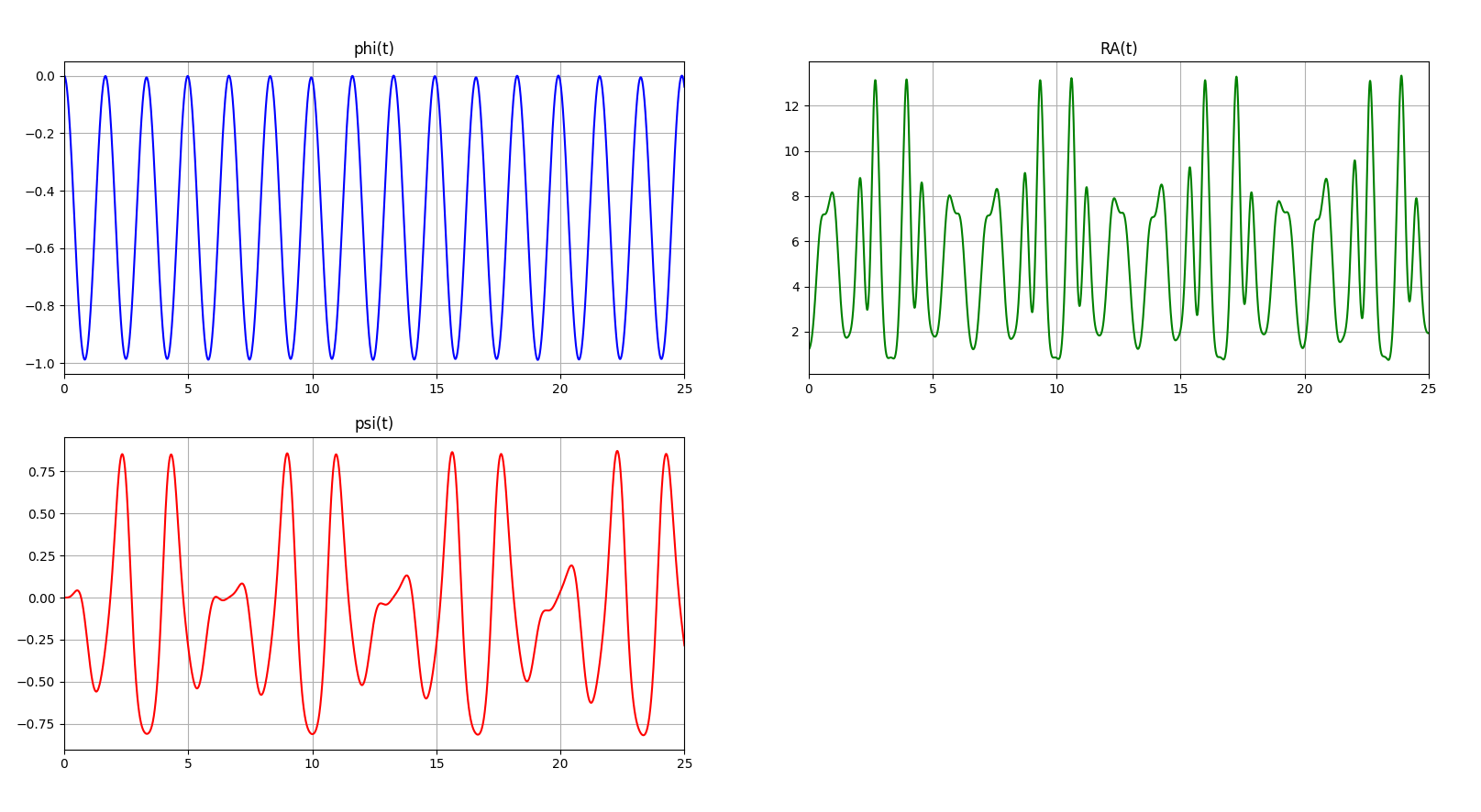


**Рисунок получившейся анимации движения:**





**Графики и сил реакции:**

****

**Код программы**

import numpy as np  
import matplotlib.pyplot as plt  
import math  
  
from matplotlib.animation import FuncAnimation  
from matplotlib.lines import Line2D  
from numpy import ndarray  
from scipy.integrate import odeint  
  
t\_fin = 25  
t = np.linspace(0, t\_fin, 2001)  
  
m1 = 50  
m2 = 0.5  
a = b = l0 = 1  
c = 250  
g = 9.8  
  
phi0 = 0  
psi0 = 0  
dphi0 = dpsi0 = 0  
y0 = [phi0, psi0, dphi0, dpsi0]  
  
  
def rotate\_2D(X, Y, angle) -> tuple:  
 rotated\_X = X \* np.cos(angle) - Y \* np.sin(angle)  
 rotated\_Y = X \* np.sin(angle) + Y \* np.cos(angle)  
  
 return rotated\_X, rotated\_Y  
  
  
def odesys(y, t, m1, m2, a, b, l0, c, g) -> ndarray:  
 l = ((8 \* (a \*\* 2) \* (1 - np.cos(y[0]))) + (l0 \* (l0 - 4 \* a \* np.sin(y[0])))) \*\* 0.5  
 dy = np.zeros(4)  
 dy[0] = y[2]  
 dy[1] = y[3]  
  
 a11 = a \* ((4 / 3) \* m1 + m2)  
 a12 = m2 \* b \* np.sin(y[1] - y[0])  
 a21 = a \* np.sin(y[1] - y[0])  
 a22 = b  
  
 b1 = c \* ((l0 / l) - 1) \* (4 \* a \* np.sin(y[0]) - 2 \* l0 \* np.cos(y[0])) - \  
 ((m1 + m2) \* g \* np.cos(y[0])) - (m2 \* b \* ((y[3]) \*\* 2) \* np.cos(y[1] - y[0]))  
  
 b2 = (a \* ((y[3]) \*\* 2) \* np.cos(y[1] - y[0])) - (g \* np.sin(y[1]))  
  
 dy[2] = (b1 \* a22 - b2 \* a12) / (a11 \* a22 - a12 \* a21)  
 dy[3] = (b2 \* a11 - b1 \* a21) / (a11 \* a22 - a12 \* a21)  
  
 return dy  
  
  
Y = odeint(odesys, y0, t, (m1, m2, a, b, l0, c, g))  
  
phi = Y[:, 0]  
ksi = Y[:, 1]  
dphi = Y[:, 2]  
dpsi = Y[:, 3]  
ddphi = [odesys(y, t, m1, m2, a, b, l0, c, g)[2] for y, t in zip(Y, t)]  
ddpsi = [odesys(y, t, m1, m2, a, b, l0, c, g)[3] for y, t in zip(Y, t)]  
  
RA = m2 \* (g \* np.cos(ksi) + b \* (dpsi \*\* 2) + a \* (ddphi \* np.cos(ksi - phi) + (dphi \*\* 2) \* np.sin(ksi - phi)))  
  
fig\_for\_graphs = plt.figure(figsize=[13, 7])  
  
ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 1)  
ax\_for\_graphs.plot(t, phi, color='Blue')  
ax\_for\_graphs.set\_title("phi(t)")  
ax\_for\_graphs.set(xlim=[0, t\_fin])  
ax\_for\_graphs.grid(True)  
  
ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 3)  
ax\_for\_graphs.plot(t, ksi, color='Red')  
ax\_for\_graphs.set\_title("psi(t)")  
ax\_for\_graphs.set(xlim=[0, t\_fin])  
ax\_for\_graphs.grid(True)  
  
ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 2, 2)  
ax\_for\_graphs.plot(t, RA, color='Green')  
ax\_for\_graphs.set\_title("RA(t)")  
ax\_for\_graphs.set(xlim=[0, t\_fin])  
ax\_for\_graphs.grid(True)  
  
a = 1  
b = a  
len\_DE = 2 \* a  
len\_AB = b  
spring\_height = 1  
D = np.array([0, 0])  
  
  
def rotate\_2D(X, Y, angle) -> tuple:  
 rotated\_X = X \* np.cos(angle) - Y \* np.sin(angle)  
 rotated\_Y = X \* np.sin(angle) + Y \* np.cos(angle)  
  
 return rotated\_X, rotated\_Y  
  
  
K = 19  
Sh = 0.4  
b = 1 / (K - 2)  
x\_spring = np.zeros(K)  
y\_spring = np.zeros(K)  
x\_spring[0] = 0  
y\_spring[0] = 0  
x\_spring[K - 1] = 1  
y\_spring[K - 1] = 0  
  
for i in range(K - 2):  
 x\_spring[i + 1] = b \* ((i + 1) - 1 / 2)  
 y\_spring[i + 1] = Sh \* (-1) \*\* i  
  
x\_E = len\_DE \* np.cos(phi)  
y\_E = len\_DE \* np.sin(phi)  
  
x\_B = len\_AB \* np.sin(ksi)  
y\_B = len\_AB \* np.cos(ksi)  
  
x\_spr = len\_DE - x\_E  
y\_spr = spring\_height - y\_E  
  
spring\_length = (x\_spr \* x\_spr + y\_spr \* y\_spr) \*\* 0.5  
  
fig = plt.figure(figsize=(9, 8))  
  
graph = fig.add\_subplot(1, 1, 1)  
graph.axis('equal')  
graph.set\_title("Model of the system movement")  
graph.set\_xlabel("Abscissa axis")  
graph.set\_ylabel("Ordinate axis")  
graph.set(xlim=[-3, 5], ylim=[-3, 5])  
  
spring\_x\_cord\_rotated, spring\_y\_cord\_rotated = rotate\_2D(x\_spring, y\_spring,  
 -(math.pi / 2 + abs(math.atan2(x\_spr[0], y\_spr[0]))))  
  
set\_spring\_after\_rotate, = graph.plot(spring\_x\_cord\_rotated + len\_DE,  
 (spring\_y\_cord\_rotated \* spring\_length[0]) + spring\_height)  
  
graph.plot(2 \* a, spring\_height, color='black', linewidth=5, marker='o')  
graph.plot([2 \* a - 0.5, 2 \* a + 0.5, 2 \* a, 2 \* a - 0.5],  
 [spring\_height + 0.7, spring\_height + 0.7, spring\_height, spring\_height + 0.7],  
 color='black', linewidth=2)  
  
graph.plot([-0.5, 0.5, 0, -0.5], [-0.5, -0.5, 0, -0.5], color='black', linewidth=2)  
graph.plot([-0.75, 0.75], [-0.5, -0.5], color='black', linewidth=3)  
  
graph.plot([0, 0], [0, 4], color='red', linewidth=3, linestyle='dashed', alpha=0.5, marker='^')  
graph.plot([0, 4], [0, 0], color='red', linewidth=3, linestyle='dashed', alpha=0.5, marker='>')  
  
draw\_DE = graph.plot(np.array([0, x\_E[0]]), np.array([0, y\_E[0]]),  
 color='green', linewidth=5, label="Beam DE", marker='o', markersize=8)[0]  
  
draw\_AB = graph.plot(np.array([x\_E[0] / 2, x\_E[0] / 2 + x\_B[0]]),  
 np.array([y\_E[0] / 2, y\_E[0] / 2 - y\_B[0]]),  
 color='blue', linewidth=5, label="Rod AB", marker='o', markersize=8)[0]  
  
graph.legend()  
  
  
def animation(i) -> Line2D:  
 draw\_DE.set\_data(np.array([0, x\_E[i]]), np.array([0, y\_E[i]]))  
 spring\_x\_cord\_rotated, spring\_y\_cord\_rotated = rotate\_2D(x\_spring \* spring\_length[i], y\_spring,  
 -(math.pi / 2 + abs(math.atan2(x\_spr[i], y\_spr[i]))))  
  
 set\_spring\_after\_rotate.set\_data(spring\_x\_cord\_rotated + len\_DE, spring\_y\_cord\_rotated + spring\_height)  
 draw\_AB.set\_data(np.array([x\_E[i] / 2, x\_E[i] / 2 + x\_B[i]]), np.array([y\_E[i] / 2, y\_E[i] / 2 - y\_B[i]]))  
  
 return draw\_DE  
  
  
show\_movement = FuncAnimation(fig, animation, frames=len(t), interval=20, repeat=True)  
animation\_running = True  
  
  
def animation\_pause(event) -> None:  
 global animation\_running  
  
 if animation\_running:  
 show\_movement.event\_source.stop()  
 animation\_running = False  
 else:  
 show\_movement.event\_source.start()  
 animation\_running = True  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 fig.canvas.mpl\_connect('button\_press\_event', animation\_pause)  
 plt.show()

**Пояснения**

В программе есть 4 основные функции:

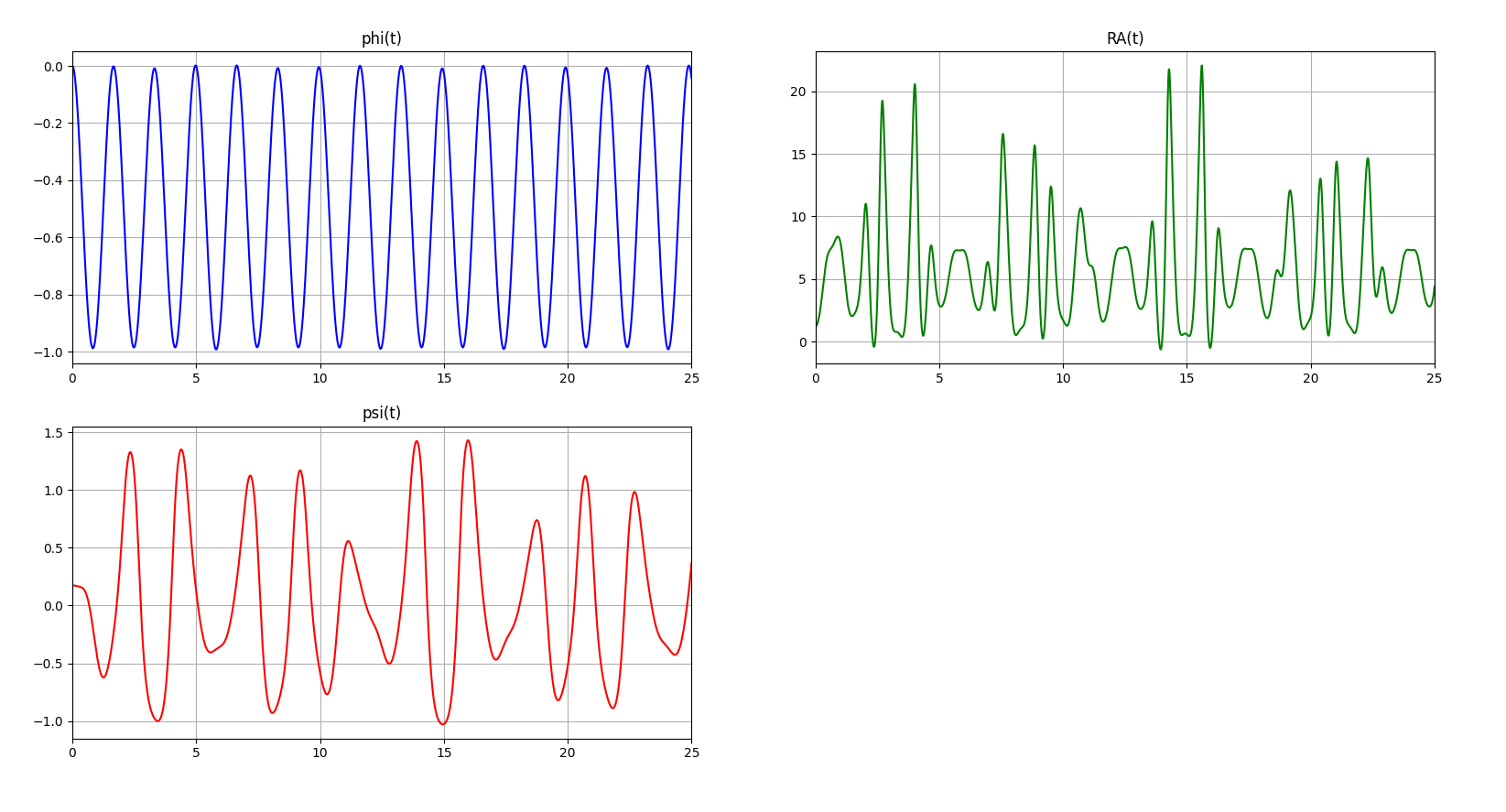
1. rotate\_2D(X, Y, angle) – выполняет поворот объекта на угол angle и возвращает измененные координаты. Поворот происходит с помощью аппарата линейной алгебра, а именно матрицы поворота.
2. animation(i) – функция смены кадра. На каждой итерации меняет положение балки и стержня, а также изменят состояние пружины (ее угол поворота и растяжение).
3. animation\_pause(event) – ставит анимацию на паузу, если нажать кнопку мыши.
4. odesys(y, t, m1, m2, a, b, l0, c, g) – функция для метода odesys, который решает дифференциальные уравнения

**Примеры работы:**

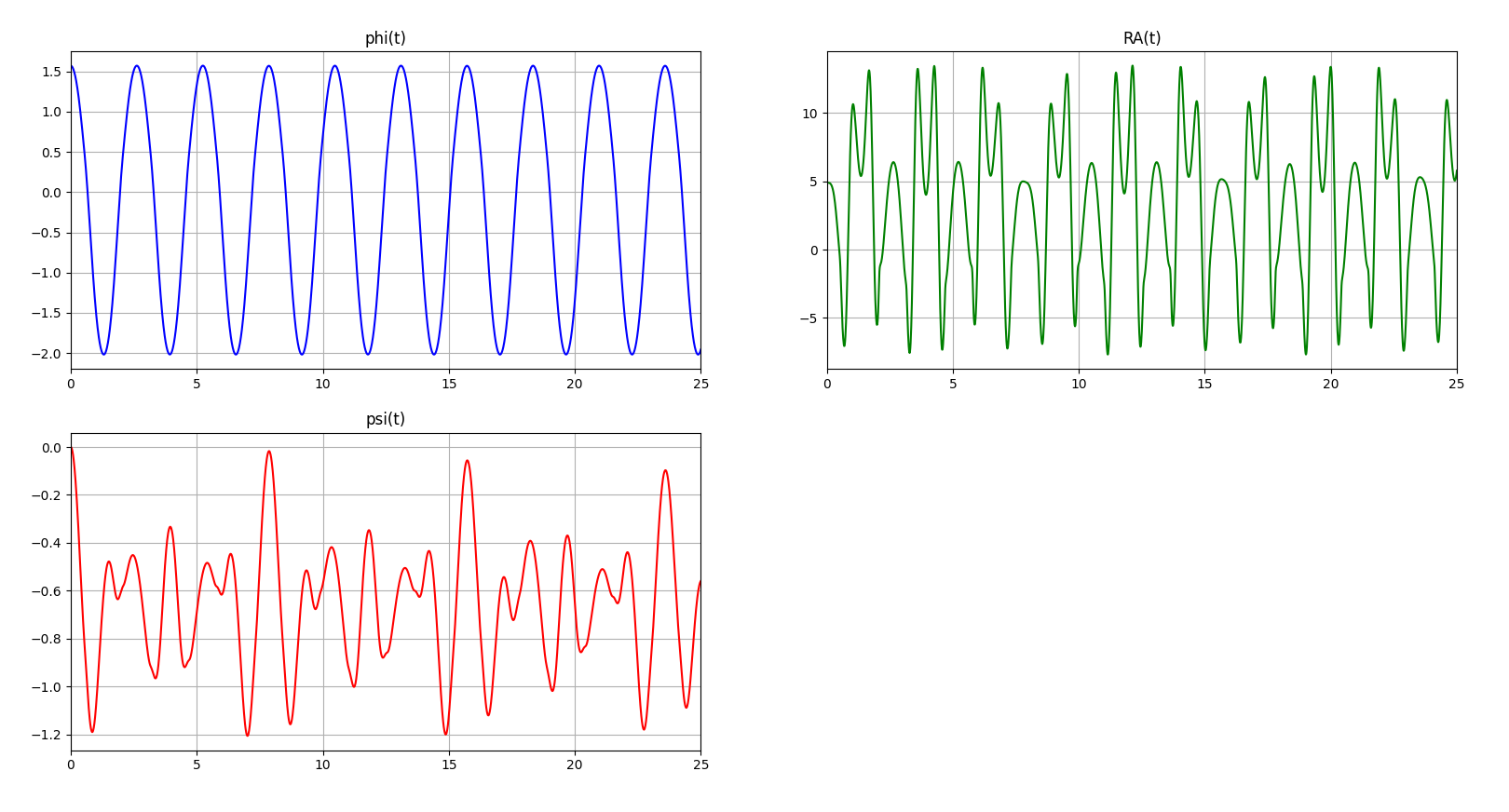
Приведу несколько примеров работы моей программы.

1. Начальные условия из учебника

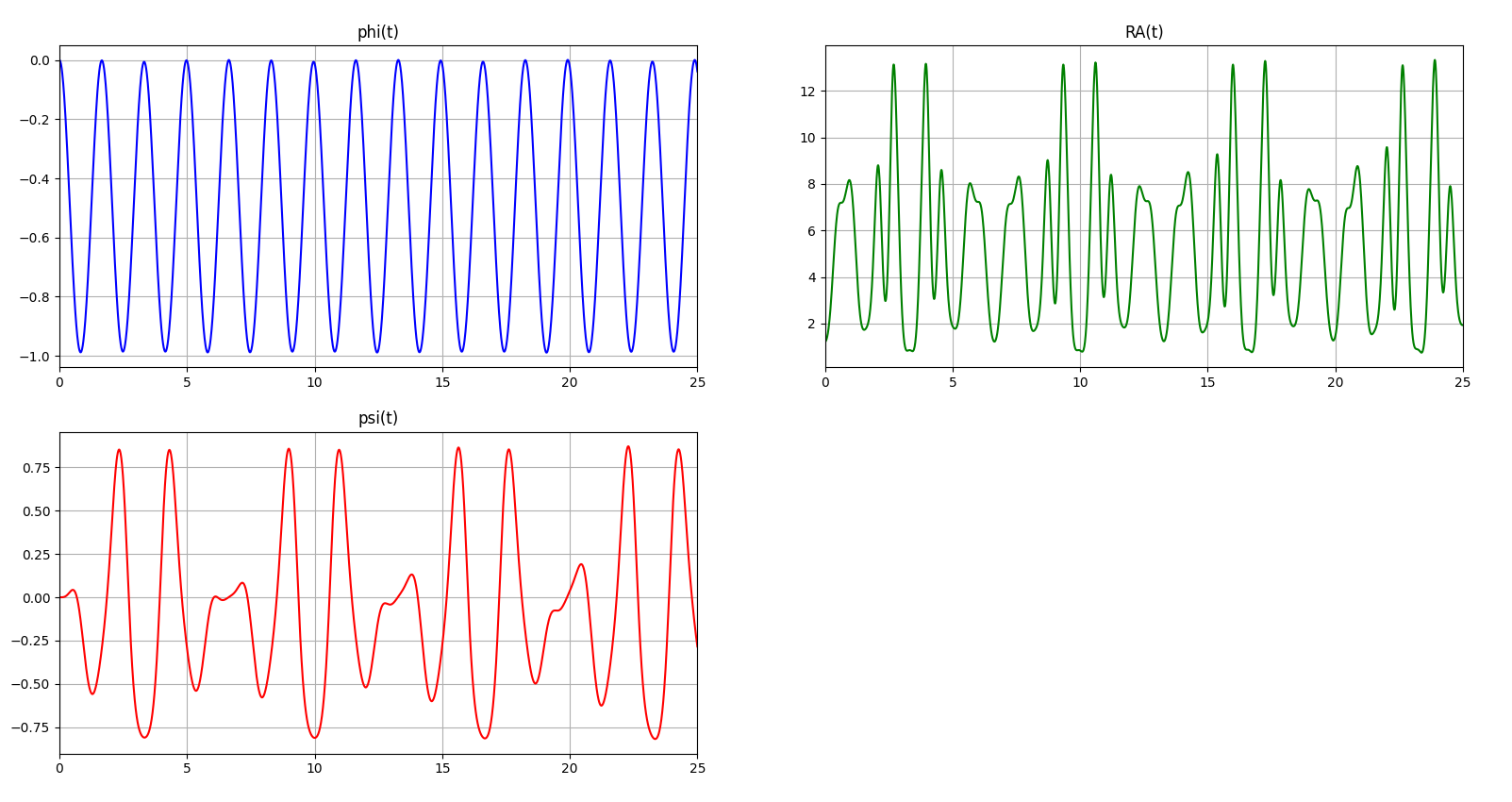
m1 = 50 kg, m2 = 0.5 kg, a = b = l0 = 1 m, c = 250 N/m, t0 = 0, 0 = 0, 0 = /18, 0 = 0 = 0

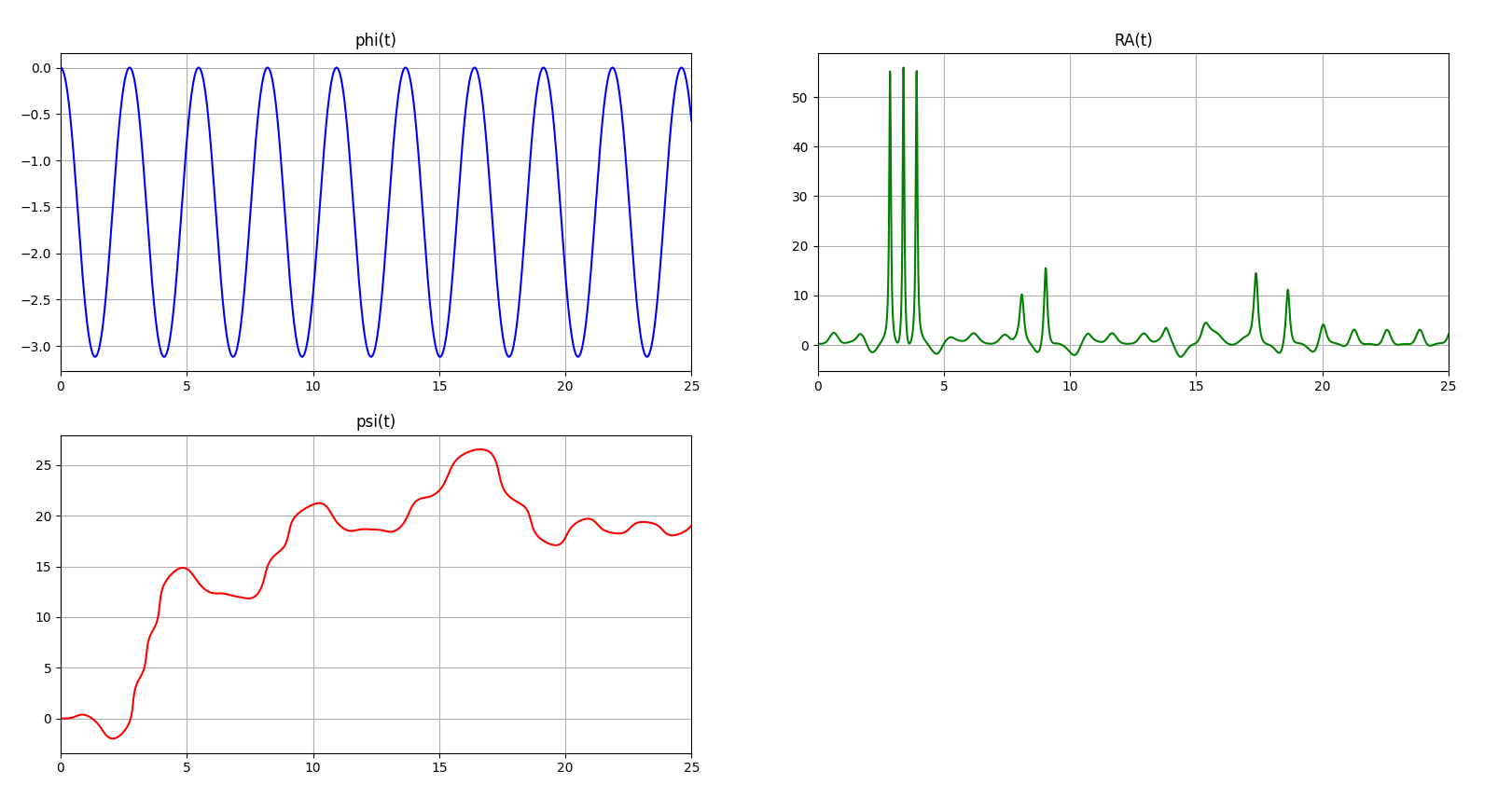


1. Все значения из учебника, кроме начального положения балки и начального угла стержня, он равен 0. Балка начинает движение с угла



1. Все значения нулевые



1. Масса балки 5000 кг, масса груза 0.1 кг

**Вывод**

Я успешно выполнил лабораторную работу по теоретической механике. С помощью языка программирования Python и библиотек matplotlib и numpy я схематично проанимировал движение балки, прикрепленной к пружине, и невесомого стержня с грузом на конце, который прикреплен к середине балки.

Благодаря этой лабораторной работе, я научился работать с 2D анимацией в matplotlib, иллюстрировать движение системы с помощью Python.

В моей программе используются реальные законы движения, благодаря чему можно посмотреть, как эта система будет вести себя в реальной жизни.