Московский авиационный институт

(национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

**Лабораторная работа №3**

**по курсу «Теоретическая механика»**

**Составление и численное решения дифференциальных уравнений движения системы и ее анимация.**

Выполнил студент группы М8О-206Б-20

Семин Александр Витальевич

Преподаватель: Сухов Е. А.

Дата: 16.12.21

Оценка:

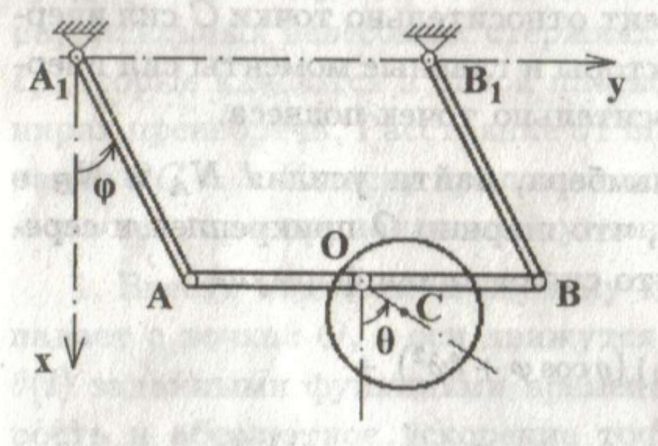
Москва, 2021

**Вариант №21**

**Задание:**

Необходимо составить и численно решить дифференциальные уравнения движения системы (уравнения Лагранжа второго рода), а затем реализовать анимацию движения механической системы используя язык программирования Python.

**Механическая система:**



Текст программы:

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from scipy.integrate import odeint

import sympy as sp

import math

def Circle(X, Y):

CX = [X + 0.75 \* math.cos(i/100) for i in range(0, 628)]

CY = [Y + 0.75 \* math.sin(i/100) for i in range(0, 628)]

return CX, CY

def anima(i):

Beam\_1.set\_data([- 4, - 4 + X\_ab[i]], [0, Y\_ab[i]])

Beam\_2.set\_data([4, 4 + X\_ab[i]], [0, Y\_ab[i]])

Beam\_3.set\_data([- 4 + X\_ab[i], 4 + X\_ab[i]], [Y\_ab[i], Y\_ab[i]])

Beam\_4.set\_data([X\_ab[i], X\_ab[i] + X\_o[i]], [Y\_ab[i], Y\_ab[i] + Y\_o[i]])

circle.set\_data(\*Circle(X\_ab[i] + X\_o[i], Y\_ab[i] + Y\_o[i]))

return Beam\_1, Beam\_2, Beam\_3, Beam\_4, circle,

def formY(y, t, f\_om\_tetta, f\_om\_phi):

y1, y2, y3, y4 = y

dydt = [y4, y3, f\_om\_tetta(y1, y2, y3, y4), f\_om\_phi(y1, y2, y3, y4)]

return dydt

beam\_length = 4

beam\_length\_2 = 1.5

m\_2 = 4

m\_1 = 2

g = 10

l = 4

b = 0.5

k = 0.2

R = 0.75

t = sp.Symbol('t')

phi = sp.Function('phi')(t)

tetta = sp.Function('tetta')(t)

om = sp.Function('om')(t)

om\_2 = sp.Function('om\_2')(t)

E\_ab = (m\_2 \* om\*\*2 \* l\*\*2) / 2

Vx\_c = om \* l \* sp.sin(phi)

Vy\_c = om \* l \* sp.cos(phi)

Vx\_o = om\_2 \* sp.sin(tetta) \* b

Vy\_o = om\_2 \* sp.cos(tetta) \* b

J\_cir = (m\_1 \* R\*\*2) / 2

E\_cir = m\_1 \* ((Vx\_o + Vx\_c)\*\*2 + (Vy\_o + Vy\_c)\*\*2) / 2 + (J\_cir \* om\_2\*\*2)/2

Ekin = E\_ab + E\_cir

Pi1 = m\_2 \* g \* (l \* (1 - sp.cos(phi)) + b)

Pi2 = m\_1 \* g \* (l \* (1 - sp.cos(phi)) + b \* (1 - sp.cos(tetta)))

Epot = Pi1 + Pi2

M = - k \* om\_2

L = Ekin - Epot

*#equations*

ur1 = sp.diff(sp.diff(L, om\_2), t) - sp.diff(L, tetta) - M

ur2 = sp.diff(sp.diff(L, om), t) - sp.diff(L, phi)

*#isolating second derivatives(dV/dt and dom/dt) using Kramer's method*

a11 = ur1.coeff(sp.diff(om\_2, t), 1)

a12 = ur1.coeff(sp.diff(om, t), 1)

a21 = ur2.coeff(sp.diff(om\_2, t), 1)

a22 = ur2.coeff(sp.diff(om, t), 1)

b1 = -(ur1.coeff(sp.diff(om\_2, t), 0)).coeff(sp.diff(om, t), 0).subs([(sp.diff(tetta, t), om\_2), (sp.diff(phi, t), om)])

b2 = -(ur2.coeff(sp.diff(om\_2, t), 0)).coeff(sp.diff(om, t), 0).subs([(sp.diff(tetta, t), om\_2), (sp.diff(phi, t), om)])

detA = a11\*a22-a12\*a21

detA1 = b1\*a22-b2\*a21

detA2 = a11\*b2-b1\*a21

dom\_tettadt = detA2/detA

dom\_phidt = detA1/detA

countOfFrames = 100

*# Constructing the system of differential equations*

T = np.linspace(0, 10, countOfFrames)

*# Pay attention here, the function lambdify translate function from the sympy to numpy and then form arrays much more*

*# faster then we did using subs in previous lessons!*

f\_om\_tetta = sp.lambdify([phi, tetta, om\_2, om], dom\_tettadt, "numpy")

f\_om\_phi = sp.lambdify([phi, tetta, om\_2, om], dom\_phidt, "numpy")

y0 = [0.75, 0.75, 15, 1]

sol = odeint(formY, y0, T, args = (f\_om\_phi, f\_om\_tetta))

X\_ab\_func = sp.lambdify(phi, sp.sin(phi) \* l)

Y\_ab\_func = sp.lambdify(phi, - sp.cos(phi) \* l)

X\_o\_func = sp.lambdify(tetta, sp.sin(tetta) \* b)

Y\_o\_func = sp.lambdify(tetta, - sp.cos(tetta) \* b)

*#constructing corresponding arrays (Very fast, thanks to lambdify)*

X\_ab = X\_ab\_func(sol[:, 0])

Y\_ab = Y\_ab\_func(sol[:, 0])

Phi = sol[:, 1]

X\_o = X\_o\_func(sol[:, 1])

Y\_o = Y\_o\_func(sol[:, 1])

fig = plt.figure(figsize = (17, 8))

ax1 = fig.add\_subplot(1, 2, 1)

ax1.axis('equal')

ax1.set(xlim = [-10, 10], ylim = [-10, 10])

ax2 = fig.add\_subplot(4, 2, 2)

ax2.plot(T, sol[:, 3])

ax2.set\_xlabel('T')

ax2.set\_ylabel('Om\_phi')

ax3 = fig.add\_subplot(4, 2, 4)

ax3.plot(T, sol[:, 2])

ax3.set\_xlabel('T')

ax3.set\_ylabel('Om\_tetta')

Beam\_1, = ax1.plot([-2, -2 + X\_ab[0]], [0, Y\_ab[0]], 'black')

Beam\_2, = ax1.plot([2, 2 + X\_ab[0]], [0, Y\_ab[0]], 'black')

Beam\_3, = ax1.plot([- 4 + X\_ab[0], 4 + X\_ab[0]], [Y\_ab[0], Y\_ab[0]], 'black')

Beam\_4, = ax1.plot([X\_ab[0], X\_ab[0] + X\_o[0]], [Y\_ab[0], Y\_ab[0] + Y\_o[0]], 'black')

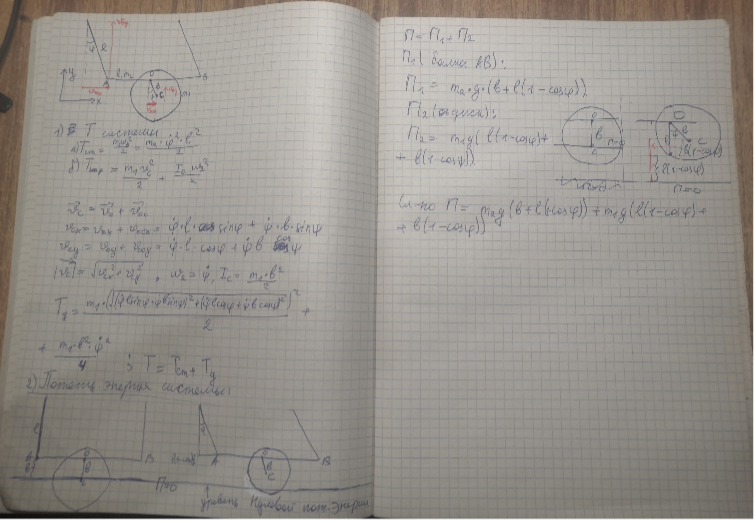
circle, = ax1.plot(\*Circle(X\_ab[0] + X\_o[0], Y\_ab[0] + Y\_o[0]), 'black')

anim = FuncAnimation(fig, anima, frames=countOfFrames, interval=100, blit=True)

plt.show()

**Результат работы:**

|  |
| --- |
|  |
|  |

**Вывод формул:**

**Вывод:**

В ходе выполнения работы я столкнулся с решением уравнений Лагранжа второго рода и, освежив знания по их решению, написал программу для анимации системы своего варианта. Уравнения Лагранжа второго рода являются хорошим способом составления уравнений движения механических систем.