Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Информационные технологии и прикладная математика»

Лабораторная работа №2
по курсу «Теоретическая механика»
Анимация системы

Выполнил студент группы М8О-215Б-23
Беличенко Михаил Валериевич
Преподаватель: Беличенко Михаил Валериевич

Оценка: восхитительно!
Дата: 21.12.24

Москва, 2021

Вариант № 8

Задание:

Реализовать анимацию движения механической системы в среде Octave (или Matlab)

Механическая система:

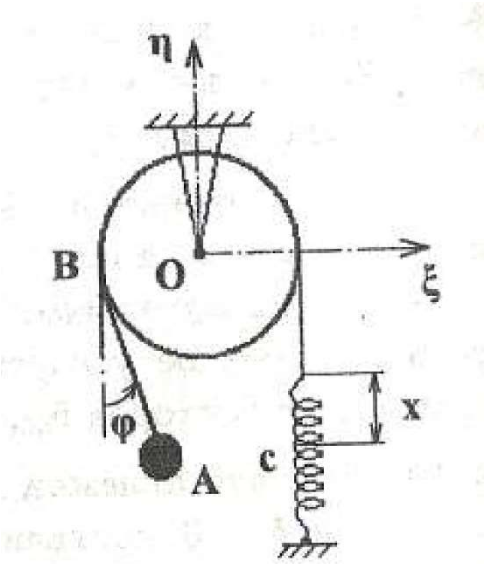

Рис. 8

Текст программы

```
# 0. Импортируем необходимые библиотеки
import math
import numpy as np  # numpy для математических вычислений
import matplotlib.pyplot as plot  # для создания графиков
from matplotlib.animation import FuncAnimation  # для анимации

# 1. Создаем фигуру и оси для графика
fgr = plot.figure(figsize=(15, 10))
gs = fgr.add_gridspec(4, 2)
gr = fgr.add_subplot(gs[:, 0])  # механизм занимает всю левую часть
gr.axis('equal')

# 2. Задаем основные параметры
STEPS = 500
STARBT_VALUE = 0
END_VALUE = 4 * math.pi
XO = 3  # x-координата центра блока
YO = 4  # y-координата центра блока
RB = 0.5  # радиус блока
Y0 = 2.3  # начальная амплитуда
RS = 0.1  # радиус малого круга
SA = np.pi / 18  # Начальное отклонение
NP = 20  # количество витков пружины

# 3. Создаем временный массив
t = np.linspace(STARBT_VALUE, END_VALUE, STEPS)  # временной массив от 0 до
2п
```

```
y = np.sin(t)  # синусоидальное движение

# 4. Рисуем неподвижные части механизма
gr.plot([2, 4], [0, 0], 'black', linewidth=3)  # нижняя опора
gr.plot([2, 4], [YO + 0.7, YO + 0.7], 'black', linewidth=3)  # верхняя опора
gr.plot([XO - 0.1, XO, XO + 0.1], [YO + 0.7, YO, YO + 0.7], 'black')  #
крепление

# 5. Расчет координат для движущихся частей
y_l = y + Y0  # левая часть механизма
y_r = y - Y0  # правая часть механизма
phi = SA * np.sin(2 * t)  # угол поворота. 2t так как груз совершает два
полных колебания

# Координаты точек механизма
Xb = XO - RB  # x-координата точки B
Yb = YO  # y-координата точки B

# Координаты точки A (движущаяся точка) (переход из ПСК в ДСК)
Xa = Xb + y_l * np.sin(phi)
Ya = Yb - y_l * np.cos(phi)

# 6. Создаем начальные элементы анимации
AB = gr.plot([Xa[0], Xb], [Ya[0], Yb], 'green')[0]  # стержень AB
L = gr.plot([XO + RB, XO + RB], [YO, YO + y_r[0]], 'green')[0]  #
вертикальный стержень

# 7. Создаем круг
Alp = np.linspace(0, 2 * np.pi, 100)  # углы для построения окружности
Xc = np.cos(Alp)  # x-координаты точек окружности
Yc = np.sin(Alp)  # y-координаты точек окружности

# Рисуем блок и малый круг
Block = gr.plot(XO + RB * Xc, YO + RB * Yc, 'black')[0]  # основной блок
m = gr.plot(Xa[0] + RS * Xc, Ya[0] + RS * Yc, 'black')[0]  # малый круг

# Создаем пружину
Yp = np.linspace(0, 1, 2 * NP + 1)  # y-координаты точек пружины
Xp = 0.15 * np.sin(np.pi / 2 * np.arange(2 * NP + 1))  # x-координаты точек
пружины
Pruzh = gr.plot(XO + RB + Xp, (YO + y_r[0]) * Yp)[0]  # рисуем пружину


# 8. Создаем графики
vx_plot = fgr.add_subplot(gs[0, 1])  # скорость по x
vy_plot = fgr.add_subplot(gs[1, 1])  # скорость по y
ax_plot = fgr.add_subplot(gs[2, 1])  # ускорение по x
ay_plot = fgr.add_subplot(gs[3, 1])  # ускорение по y

# Добавляем расчет скоростей и ускорений
dt = (END_VALUE - STARBT_VALUE) / STEPS
# Скорости (используем центральные разности)
vx = np.gradient(Xa, dt)
vy = np.gradient(Ya, dt)
# Ускорения
ax = np.gradient(vx, dt)
ay = np.gradient(vy, dt)

# Создаем временную ось для графиков
time_array = np.linspace(0, END_VALUE, STEPS)

# Инициализация линий для графиков
vx_line, = vx_plot.plot([], [], 'b-', label='Vx')
vy_line, = vy_plot.plot([], [], 'r-', label='Vy')
```

```
ax_line, = ax_plot.plot([], [], 'g-', label='Ax')
ay_line, = ay_plot.plot([], [], 'm-', label='Ay')

# Настройка графиков
for subplot in [vx_plot, vy_plot, ax_plot, ay_plot]:
    subplot.grid(True)
    subplot.legend()
    subplot.set_xlim(0, END_VALUE)

vx_plot.set_ylabel('Скорость по X')
vy_plot.set_ylabel('Скорость по Y')
ax_plot.set_ylabel('Ускорение по X')
ay_plot.set_ylabel('Ускорение по Y')
ax_plot.set_xlabel('Время')
ay_plot.set_xlabel('Время')

# Установка пределов для графиков
vx_plot.set_ylim(np.min(vx)*1.1, np.max(vx)*1.1)
vy_plot.set_ylim(np.min(vy)*1.1, np.max(vy)*1.1)
ax_plot.set_ylim(np.min(ax)*1.1, np.max(ax)*1.1)
ay_plot.set_ylim(np.min(ay)*1.1, np.max(ay)*1.1)


# 9. Функция обновления кадров анимации
def run(i):
    # Обновляем механизм
    m.set_data([Xa[i] + RS * Xc], [Ya[i] + RS * Yc])
    AB.set_data([Xa[i], Xb], [Ya[i], Yb])
    L.set_data([XO + RB, XO + RB], [YO, YO + y_r[i]])
    Pruzh.set_data(XO + RB + Xp, (YO + y_r[i]) * Yp)

    # Обновляем графики
    vx_line.set_data(time_array[:i], vx[:i])
    vy_line.set_data(time_array[:i], vy[:i])
    ax_line.set_data(time_array[:i], ax[:i])
    ay_line.set_data(time_array[:i], ay[:i])

    return [m, AB, Block, Pruzh, vx_line, vy_line, ax_line, ay_line]


# 10. Создаем анимацию
anim = FuncAnimation(fgr, run, frames=STEPS, interval=1)  # interval=1 задает
скорость анимации

# Показываем результат
plot.tight_layout()
plot.show()
```

Результат работы: