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-207Б-20
Чекменев Вячеслав Алексеевич
Преподаватель: Чекина Евгения Алексеевна

Оценка:

Дата:

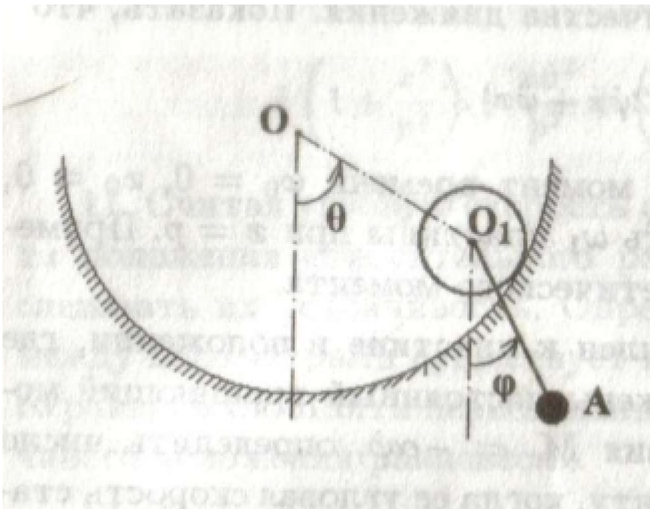
Москва, 2021

Вариант №«25»

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 formY(y, t, fomega_thetta, fOm):
    y1, y2, y3, y4 = y
    dydt = [y3, y4, fomega_thetta(y1, y2, y3, y4), fOm(y1, y2, y3, y4)]
    return dydt

Steps = 1001
R_Ground = 6
R_Circle = R_Ground/6
m1 = 0.0001
m2 = 0.00005
g = 9.81
l = R_Ground/2 # length of the palka between O1 and A

# defining t as a symbol (it will be the independent variable)
t = sp.Symbol('t')

# defining s, phi, V=ds/dt and om=dphi/dt as functions of 't'
thetta = sp.Function('thetta')(t)  # s
phi = sp.Function('phi')(t)  # phi
omega_thetta = sp.Function('omega_thetta')(t)  # V
omega_phi = sp.Function('omega_phi')(t)  # om
# Check the derivating process
print(sp.diff(5*omega_thetta**2, omega_thetta))

#1 defining the kinetic energy
V_O1 = (R_Ground - R_Circle) * omega_thetta
om1 = V_O1 / R_Circle
J_O1 = (m1 * R_Circle**2) / 2

T1 = (m1 * V_O1**2) / 2 + (J_O1 * om1**2) / 2
Ve = V_O1
Vr = omega_phi * l # changed sp.diff(phi, t) to omega_phi
T2 = (m2 * (Ve**2 + Vr**2 + 2*Ve*Vr*sp.cos(thetta - phi))) / 2
```

```
T = T1 + T2

#2 defining the potential energy
P1 = -m1*g*(R_Ground - R_Circle)*sp.cos(thetta)
P2 = -m2*g*((R_Ground - R_Circle)*sp.cos(thetta) + l*sp.cos(phi))

P = P1 + P2

#Lagrange function
L = T - P

#equations
ur1 = sp.diff(sp.diff(L,omega_thetta),t)-sp.diff(L,thetta)
ur2 = sp.diff(sp.diff(L,omega_phi),t)-sp.diff(L,phi)

# isolating second derivatives(dV/dt and dom/dt) using Kramer's method
a11 = ur1.coeff(sp.diff(omega_thetta, t), 1)
a12 = ur1.coeff(sp.diff(omega_phi, t), 1)
a21 = ur2.coeff(sp.diff(omega_thetta, t), 1)
a22 = ur2.coeff(sp.diff(omega_phi, t), 1)
b1 = -(ur1.coeff(sp.diff(omega_thetta, t), 0)).coeff(sp.diff(omega_phi, t),
                                                     0).subs([(sp.diff(thetta, t),
omega_thetta), (sp.diff(phi, t), omega_phi)])
b2 = -(ur2.coeff(sp.diff(omega_thetta, t), 0)).coeff(sp.diff(omega_phi, t),
                                                     0).subs([(sp.diff(thetta, t),
omega_thetta), (sp.diff(phi, t), omega_phi)])

detA = a11*a22-a12*a21
detA1 = b1*a22-b2*a21
detA2 = a11*b2-b1*a21

domega_thettadt = detA1/detA
domega_phidt = detA2/detA

# Constructing the system of differential equations
T = np.linspace(0, 12, Steps)
fomega_thetta = sp.lambdify([thetta, phi, omega_thetta, omega_phi], domega_thettadt,
"numpy")
fomega_phi = sp.lambdify([thetta, phi, omega_thetta, omega_phi], domega_phidt,
"numpy")
y0 = [-1, 0, 0.1, -0.1] ################################################
sol = odeint(formY, y0, T, args=(fomega_thetta, fomega_phi))


Thetta = sol[:, 0]
Phi = sol[:, 1]
Omega_thetta = sol[:, 2]
Omega_phi = sol[:, 3]

# static

# Point O
X_O = R_Ground
Y_O = R_Ground

# Ground
alpha = np.linspace(-math.pi, 0, 500)

X_Ground = R_Ground + R_Ground * np.cos(alpha)
Y_Ground = R_Ground + R_Ground * np.sin(alpha)

# circle
beta = np.linspace(0, 2*math.pi, 500)

X_Circle = R_Circle * np.cos(beta)
Y_Circle = R_Circle * np.sin(beta)

# constructing functions
# Point O1
x_o1 = X_O + (R_Ground - R_Circle) * sp.sin(thetta)
```

```
y_o1 = Y_O - (R_Ground - R_Circle) * sp.cos(thetta)
X_O1 = sp.lambdify(thetta, x_o1)
Y_O1 = sp.lambdify(thetta, y_o1)

# point A
X_A = sp.lambdify([thetta, phi], x_o1 + l*sp.sin(phi))
Y_A = sp.lambdify([thetta, phi], y_o1 - l*sp.cos(phi))

XO1 = X_O1(sol[:, 0])
YO1 = Y_O1(sol[:, 0])

# Points C1 and C2 -- points on surface of the circle relative to point O1
X_C1 = sp.lambdify([thetta], x_o1 + R_Circle*sp.sin(thetta))
X_C2 = sp.lambdify([thetta], x_o1 - R_Circle*sp.sin(thetta))
Y_C1 = sp.lambdify([thetta], y_o1 + R_Circle*sp.cos(thetta))
Y_C2 = sp.lambdify([thetta], y_o1 - R_Circle*sp.cos(thetta))
XC1 = X_C1(sol[:, 0])
XC2 = X_C2(sol[:, 0])
YC1 = Y_C1(sol[:, 0])
YC2 = Y_C2(sol[:, 0])

XA = X_A(sol[:, 0], sol[:, 1])
YA = Y_A(sol[:, 0], sol[:, 1])

# some settings
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.axis("equal")
ax.set(xlim=(0, 12), ylim=(0, 12))

# plot zero state
Ground = ax.plot(X_Ground, Y_Ground, color='black', linewidth=2)
Point_O = ax.plot(X_O, Y_O, color='red', linewidth=4)
Draw_palka = ax.plot([X_O, XO1[0]], [Y_O, YO1[0]], 'r--')[0]
Draw_palka1 = ax.plot([XC1[0], XC2[0]], [YC1[0], YC2[0]], 'b')[0]

Draw_Circle = ax.plot(
    X_Circle + XO1[0], Y_Circle + YO1[0], color='blue', linewidth=1)[0]
Draw_point_O1 = ax.plot(XO1[0], YO1[0], color='blue',
                        linewidth=3, marker='o')[0]
Draw_point_A = ax.plot(XA[0], YA[0], 'r', marker='o', markersize=15)[0]
Draw_palka_O1_A = ax.plot([XO1[0], XA[0]], [YO1[0], YA[0]], 'b')[0]

# graphs
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, 12])
ax_for_graphs.grid(True)

ax_for_graphs = fig_for_graphs.add_subplot(2, 2, 2)
ax_for_graphs.plot(T, Thetta, color='red')
ax_for_graphs.set_title('Thetta(t)')
ax_for_graphs.set(xlim=[0, 12])
ax_for_graphs.grid(True)

ax_for_graphs = fig_for_graphs.add_subplot(2,2,3)
ax_for_graphs.plot(T, Omega_phi, color='green')
ax_for_graphs.set_title("phi'(t) = omega_phi(t)")
ax_for_graphs.set(xlim=[0, 12])
ax_for_graphs.grid(True)

ax_for_graphs = fig_for_graphs.add_subplot(2, 2, 4)
ax_for_graphs.plot(T, Omega_thetta, color='black')
ax_for_graphs.set_title("thetta'(t) = omega_thetta(t)")
ax_for_graphs.set(xlim=[0, 12])
ax_for_graphs.grid(True)

# function for updating state of the system
def kinoteatr_five_zvezd_na_novokuzneckoy(i):
```

```
    Draw_point_O1.set_data(XO1[i], YO1[i])
    Draw_Circle.set_data(X_Circle + XO1[i], Y_Circle + YO1[i])
    Draw_palka.set_data([X_O, XO1[i]], [Y_O, YO1[i]])
    Draw_palka1.set_data([XC1[i], XC2[i]], [YC1[i], YC2[i]])
    Draw_point_A.set_data(XA[i], YA[i])
    Draw_palka_O1_A.set_data([XO1[i], XA[i]], [YO1[i], YA[i]])
    return [Draw_point_O1, Draw_Circle, Draw_palka, Draw_point_A, Draw_palka1]


anime = FuncAnimation(fig, kinoteatr_five_zvezd_na_novokuzneckoy,
                      frames=Steps, interval=.5)

plt.show()
```

Результат работы: