Московский авиационный институт

(национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

**Лабораторная работа №3**

**по курсу «Теоретическая механика»**

**Анимация системы**

Выполнил студент группы М8О-201Б-20

Сазонов Вадим Кириллович

Оценка:

Дата:

Москва, 2021

**Вариант №12«Динамика Системы»**

**Задание:**

проинтегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы при помощи средств Python. Построить анимацию движения системы, а также графики законов движения системы. А также составить уравнение Лагранжа второго рода для заданной системы.

Расчет уравнения Лагранжа производится по следующему алгоритму:

1) Определить количество степеней свободы системы 𝑛.

2) В соответствии с количеством степеней свободы системы, полученным в п. 1, ввести соответствующее число обобщенных координат (𝑞1,𝑞2,…,𝑞𝑛).

3) Посчитать кинетическую энергию в зависимости от обобщенных координат и обобщенных скоростей 𝑇(𝑞1,…,𝑞𝑛,𝑞̇1,…,𝑞̇𝑛).

4) Найти обобщенные силы, которые соответствуют обобщенным координатам (𝑄1,…,𝑄𝑛).

5) Составить 𝑛 уравнений Лагранжа второго рода. Общий вид -го уравнения Лагранжа второго рода выглядит следующим образом:

𝑑dt (𝜕𝑇/∂q̇i)−𝜕𝑇/∂qi =𝑄𝑖.

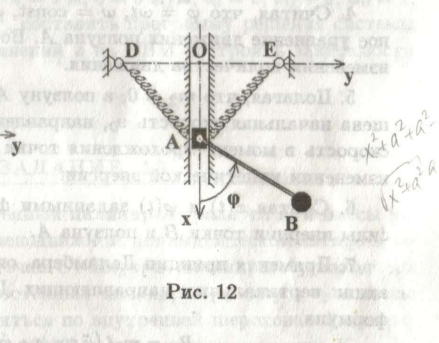
Если система консервативна, то можно упростить вид уравнения, используя функцию Лагранжа

𝐿=𝑇−𝛱.

Тогда -е уравнение Лагранжа второго рода для консервативной системы будет выглядеть следующим образом:

𝑑𝑑𝑡(𝜕𝐿/∂q̇i)−𝜕𝐿/∂qi =0.

**Механическая система:**



**Текст программы**

Основная :

import sympy as sp

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.animation import FuncAnimation

from matplotlib.patches import Rectangle

from matplotlib.patches import Circle

import math

from scipy.integrate import odeint

def formY(y, t, fv, fw):

y1, y2, y3, y4 = y

dydt = [y3, y4, fv(y1, y2, y3, y4), fw(y1, y2, y3, y4)]

return dydt

Frames = 500

Interval\_Frame = 0

Repeat\_Delay\_Anim = 0

t = sp.Symbol("t") # t is a symbol variable

x = sp.Function('x')(t) # x(t)

fi = sp.Function('fi')(t) # fi(t)

v = sp.Function('v')(t) # dx/dt, v(t)

w = sp.Function('w')(t) # dfi/dt, w(t), omega(t)

# Initializing

width = 1 # width of the rectangle

length = 2 # length of the rectangle

circle\_radius = 0.2 # radius of the circle

a = 6 # distance between spring and rectanlge (DO or OE)

m1 = 15 # mass of the rectangle

m2 = 5 # mass of the circle

g = 9.8 # const

l = 2 # length of stick

k = 10 # spring stiffness coefficient

y0 = [-5, sp.rad(45), 0, 0] # x(0), fi(0), v(0), w(0)

# Caluclting Lagrange equations

# Kinetic energy of the rectangle

Ekin1 = (m1 \* v \* v) / 2

# Squared velocity of the circle's center of mass

Vsquared = v \* v + w \* w \* l \* l - 2 \* v \* w \* l \* sp.sin(fi)

# Kinetic energy of the circle

Ekin2 = m2 \* Vsquared / 2

# Kinetic energy of system

Ekin = Ekin1 + Ekin2

# Potential energy

Spring\_delta\_x = sp.sqrt(a \* a + x \* x) - a # delta\_x^2

# We have two springs so Esprings = 2 \* (k\*delta\_x^2/2) = k\*delta\_x^2

Esprings = k \* Spring\_delta\_x \* Spring\_delta\_x

Epot = - m1 \* g \* x - m2 \* g \* (x + l \* sp.cos(fi)) + Esprings

# generalized forces

Qx = -sp.diff(Epot, x)

Qfi = -sp.diff(Epot, fi)

# Lagrange function

Lagr = Ekin - Epot

ur1 = sp.diff(sp.diff(Lagr, v), t) - sp.diff(Lagr, x)

ur2 = sp.diff(sp.diff(Lagr, w), t) - sp.diff(Lagr, fi)

print(ur1)

print()

print(ur2 / (l \* m2))

# Isolating second derivatives (dv/dt and dw/dt) using Kramer's method

a11 = ur1.coeff(sp.diff(v, t), 1)

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

a21 = ur2.coeff(sp.diff(v, t), 1)

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

b1 = -(ur1.coeff(sp.diff(v, t), 0)).coeff(sp.diff(w, t),

0).subs([(sp.diff(x, t), v), (sp.diff(fi, t), w)])

b2 = -(ur2.coeff(sp.diff(v, t), 0)).coeff(sp.diff(w, t),

0).subs([(sp.diff(x, t), v), (sp.diff(fi, t), w)])

detA = a11 \* a22 - a12 \* a21

detA1 = b1 \* a22 - b2 \* a21

detA2 = a11 \* b2 - b1 \* a21

dvdt = detA1 / detA

dwdt = detA2 / detA

# constructing the system of differential equations

T = np.linspace(0, 50, Frames)

# lambdify translates function from sympy to numpy and then form arrays faster then by using subs

fv = sp.lambdify([x, fi, v, w], dvdt, "numpy")

fw = sp.lambdify([x, fi, v, w], dwdt, "numpy")

sol = odeint(formY, y0, T, args=(fv, fw))

# sol - our solution

# sol[:,0] - x

# sol[:,1] - fi

# sol[:,2] - v (dx/dt)

# sol[:,3] - w (dfi/dt)

# point A (center of the rectangle):

ax = sp.lambdify(x, 0)

ay = sp.lambdify(x, x)

AX = ax(sol[:, 0])

AY = -ay(sol[:, 0])

# point B (center of the circle):

bx = sp.lambdify(fi, l \* sp.sin(fi))

by = sp.lambdify([x, fi], + l \* sp.cos(fi) + x)

BX = bx(sol[:, 1])

BY = -by(sol[:, 0], sol[:, 1])

# start plotting

fig = plt.figure()

ax0 = fig.add\_subplot(1, 2, 1)

ax0.axis("equal")

# constant arrays

L1X = [-width / 2, -width / 2]

L2X = [width / 2, width / 2]

LY = [min(AY) - length, max(AY) + length]

# plotting environment

ax0.plot(L1X, LY, color="grey") # left wall

ax0.plot(L2X, LY, color="grey") # right wall

sl, = ax0.plot([-a, -length / 2], [0, AY[0] + width / 2],

color="brown") # left spring (rope)

sr, = ax0.plot([a, length / 2], [0, AY[0] + width / 2],

color="brown") # right spring (rope)

ax0.plot(-a, 0, marker=".", color="black") # left joint

ax0.plot(a, 0, marker=".", color="black") # right joint

rect = plt.Rectangle((-width / 2, AY[0]), width,

length, color="black") # rectangle

circ = plt.Circle((BX[0], BY[0]), circle\_radius, color="grey") # circle

# plotting radius vector of B

R\_vector, = ax0.plot([0, BX[0]], [0, BY[0]], color="grey")

# adding statistics

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

ax2.plot(T, sol[:, 2])

ax2.set\_xlabel('t')

ax2.set\_ylabel('V')

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

ax3.plot(T, sol[:, 3])

ax3.set\_xlabel('t')

ax3.set\_ylabel('w')

plt.subplots\_adjust(wspace=0.3, hspace=0.7)

# function for initializing the positions

def init():

rect.set\_y(-length / 2)

ax0.add\_patch(rect)

circ.center = (0, 0)

ax0.add\_patch(circ)

return rect, circ

# function for recounting the positions

def anima(i):

rect.set\_y(AY[i] - length / 2)

sl.set\_data([-a, -width / 2], [0, AY[i]])

sr.set\_data([a, width / 2], [0, AY[i]])

R\_vector.set\_data([0, BX[i]], [AY[i], BY[i]])

circ.center = (BX[i], BY[i])

return sl, sr, rect, R\_vector, circ,

# animating function

anim = FuncAnimation(fig, anima, init\_func=init, frames=Frames, interval=Interval\_Frame,

blit=False, repeat=True, repeat\_delay=Repeat\_Delay\_Anim)

plt.show()

**Результат работы:**

