Московский авиационный институт

(национальный исследовательский университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Лабораторная работа № 3
по курсу "Теоретическая механика и компьютерное моделирование"

Динамика системы

Выполнил студент группы М8О-206Б-21
Лохматов Н.И
Преподаватель: Чекина Евгения Алексеевна

Оценка:
Дата: 25.12.2022

Москва, 2022

Вариант № 10

Задание:

Твёрдое тело массы M удерживается с помощью цилиндрического шарнира O и спиральной пружины жёсткости c (рис. 10). Центр тяжести тела находится в точке A, где $OA = a$, $OB = b$. Когда OA находится на вертикали, пружина не напряжена. Момент инерции тела относительно горизонтальной оси O, перпендикулярной плоскости рисунка, равен J_0. По поверхности тела движется точка D массы m, при перемещении которой возникает сила сопротивления $F = -k\,\dot{s}$, $k = const$, пропорциональная относительной скорости точки.

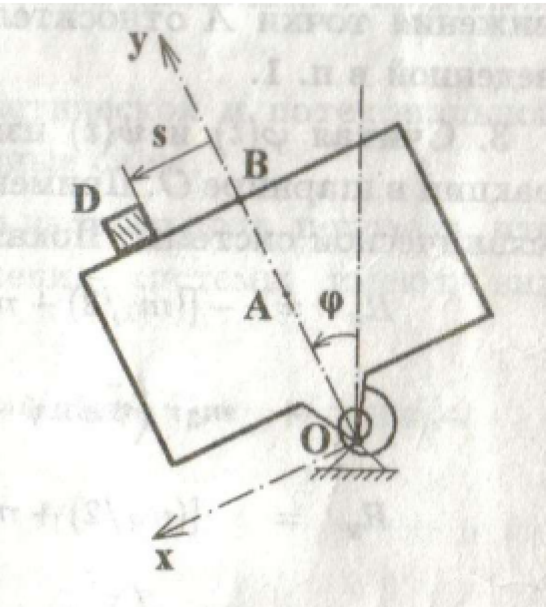

Рис. 10

1. Составить дифференциальные уравнения движения системы, исходя из общего уравнения аналитической динамики и приняв за обобщённые координаты s и ϕ.
2. Составить выражения для кинетической и потенциальной энергии системы, вычислить обобщённые силы.
3. Используя уравнения Лагранжа второго рода, показать, что дифференциальные уравнения движения системы имею вид:

$$J\,\ddot{\phi} + m(b\,\ddot{s} + 2s\,\dot{s}\dot{\phi}) = [(Ma + mb)sin\phi + mscos\phi]g - c\phi$$
$$b\,\ddot{\phi} + \ddot{s} - s\,\dot{\phi}^2 = gsin\phi - (k/m)\,\dot{s}$$
$$J = J_0 + m(b^2 + s^2)$$

Текст программы:

```
from matplotlib.patches import Rectangle
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import sympy as sp
import math
from scipy.integrate import odeint

matplotlib.use('TkAgg')


def Spring(x0, y0, phi, rad):  # return lists for a spring
    SX = [x0 + rad * t * sp.cos(t) / (2 * math.pi) for t in np.linspace(0, 2 * math.pi + (1 / 2) * math.pi +
    SY = [y0 + rad * t * sp.sin(t) / (2 * math.pi) for t in np.linspace(0, 2 * math.pi + (1 / 2) * math.pi +
    return SX, SY


def Triangle():  # на чём стоит спираль
    TX = [-1, 1, 0, -1]
    TY = [-1.5, -1.5, 0, -1.5]
    return TX, TY


def Ro(phi, rad):  # это тот зелёный лучик
    RX = [0, -rad * sp.sin(phi)]
    RY = [0, rad * sp.cos(phi)]
    return RX, RY


def Rectangle(phi, Rb, Rt):
    KX = -Rb * sp.sin(phi + 3 * math.pi / 8)  # тут точки прямоугольника
    LX = -Rt * sp.sin(phi + math.pi / 4)
    NX = -Rb * sp.sin(phi - 3 * math.pi / 8)
    MX = -Rt * sp.sin(phi - math.pi / 4)
    KY = Rb * sp.cos(phi + 3 * math.pi / 8)
    NY = Rb * sp.cos(phi - 3 * math.pi / 8)
    LY = Rt * sp.cos(phi + math.pi / 4)
    MY = Rt * sp.cos(phi - math.pi / 4)
    RcX = [KX, LX, MX, NX, KX]
    RcY = [KY, LY, MY, NY, KY]
    return RcX, RcY


def Square(phi, s):
    # сторона квадрата
    a = 1
    X1, Y1 = - math.sqrt(s ** 2 + h ** 2) * sp.sin(phi + sp.atan(s / h)), \
             math.sqrt(s ** 2 + h ** 2) * sp.cos(phi + sp.atan(s / h))
    X2, Y2 = - math.sqrt(s ** 2 + h ** 2) * sp.sin(phi + sp.atan(s / h)) - a * sp.sin(phi), \
```

```
             math.sqrt(s ** 2 + h ** 2) * sp.cos(phi + sp.atan(s / h)) + a * sp.cos(phi)
    X3, Y3 = - math.sqrt(s ** 2 + h ** 2) * sp.sin(phi + sp.atan(s / h)) - a * sp.sin(phi) - a * sp.cos(phi)
             math.sqrt(s ** 2 + h ** 2) * sp.cos(phi + sp.atan(s / h)) + a * sp.cos(phi) - a * sp.sin(phi)
    X4, Y4 = - math.sqrt(s ** 2 + h ** 2) * sp.sin(phi + sp.atan(s / h)) - a * sp.cos(phi), \
             math.sqrt(s ** 2 + h ** 2) * sp.cos(phi + sp.atan(s / h)) - a * sp.sin(phi)

    RcX = [X1, X2, X3, X4, X1]
    RcY = [Y1, Y2, Y3, Y4, Y1]
    return RcX, RcY

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

# defining parameters
m1 = 70
m2 = 20
Jo = 100
c = 10**5
k = 300
h = 6
a = 10
ro = 2.5
g = 9.81
Rtop = 6 * (2 ** 0.5)
Rbot = 3 * (5 ** 0.5)

# defining t as a symbol (it will be the independent variable)
t = sp.Symbol('t')
# here x, y, Vx, Vy, Wx, Wy, xC are functions of 't'
s = sp.Function('s')(t)
phi = sp.Function('phi')(t)
V = sp.Function('V')(t)
om = sp.Function('om')(t)

l = 1

XE = -(h + l / 2) * sp.sin(phi)
YE = (h + l / 2) * sp.cos(phi)

XD = XE - s * sp.cos(phi)
YD = YE - s * sp.sin(phi)

VxD = sp.diff(XD)
VyD = sp.diff(YD)
VmodD = sp.sqrt(VxD ** 2 + VyD ** 2)

WxD = sp.diff(VxD)
WyD = sp.diff(VyD)
```

```
WmodD = sp.sqrt(WxD ** 2 + WyD ** 2)

# constructing the Lagrange equations

# кинетическая энергия большого кубика
Tbig = (Jo * om ** 2) / 2

V2 = (VxD ** 2 + VyD ** 2).subs([(sp.diff(s, t), V), (sp.diff(phi, t), om)])

# кинетическая энергия маленького кубика
Tsmall = (m2 * V2) / 2

TT = Tbig + Tsmall

# 2 потенциальная энергия
Psmall = m2 * g * YD
Pbig = m1 * g * a * sp.cos(phi)
Pypr = (c * phi ** 2) / 2
Pi = Psmall + Pbig + Pypr

# не потенциальные силы
F = -k * sp.diff(s)

# Lagrange function
L = TT - Pi

# equations
ur1 = (sp.diff(sp.diff(L, V), t) - sp.diff(L, s) - F).simplify()
ur2 = (sp.diff(sp.diff(L, om), t) - sp.diff(L, phi)).simplify()

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

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

dVdt = detA1 / detA
domdt = detA2 / detA

T = np.linspace(0, 20, 5000)

fV = sp.lambdify([s, phi, V, om], dVdt, "numpy")
fOm = sp.lambdify([s, phi, V, om], domdt, "numpy")
y0 = [1, sp.rad(-45), 1, 1]
```

```
sol = odeint(formY, y0, T, args=(fV, f0m))

# sol - our solution
# sol[:,0] - s
# sol[:,1] - phi
# sol[:,2] - v (dx/dt)
# sol[:,3] - om (dphi/dt)

XSpr = np.zeros_like(T)
YSpr = np.zeros_like(T)

Phi = sol[:, 1]
S = sol[:, 0]

# here we start to plot
fig = plt.figure(figsize=(17, 8))

ax1 = fig.add_subplot(1, 2, 1)
ax1.axis('equal')
ax1.set(xlim=[XSpr.min() - 2 * a, XSpr.max() + 2 * a], ylim=[YSpr.min() - 2 * a, YSpr.max() + 2 * a])

# plotting a spring
SpX, SpY = Spring(XSpr[0], YSpr[0], Phi[0], a / 8)
Spr, = ax1.plot(SpX, SpY, 'black')

# plotting triangle
TrX, TrY = Triangle()
Trngl, = ax1.plot(TrX, TrY, 'brown')

# plotting Ro
RoX, RoY = Ro(Phi[0], ro)
Rorad, = ax1.plot(RoX, RoY, 'green')

# plotting Rectangle
RectangleX, RectangleY = Rectangle(Phi[0], Rbot, Rtop)
Rect, = ax1.plot(RectangleX, RectangleY, 'purple')

# plotting Square
SquareX, SquareY = Square(Phi[0], S[0])
Sqr, = ax1.plot(SquareX, SquareY, 'red')

"""constructing functions"""
countOfFrames = 200

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('Om')

plt.subplots_adjust(wspace=0.3, hspace=0.7)


# function for recounting the positions


def anima(i):
    SpX, SpY = Spring(XSpr[i], YSpr[i], Phi[i], a / 8)
    Spr.set_data(SpX, SpY)
    RoX, RoY = Ro(Phi[i], ro)
    Rorad.set_data(RoX, RoY)
    RecX, RecY = Rectangle(Phi[i], Rbot, Rtop)
    Rect.set_data(RecX, RecY)
    SqX, SqY = Square(Phi[i], S[i])
    Sqr.set_data(SqX, SqY)

    return Spr, Rorad, Rect, Sqr


# animation function


anim = FuncAnimation(fig, anima, frames=1000, interval=0.01, blit=True)

plt.show()
```

Результат работы программы: