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

Лабораторная работа №3
по курсу «Теоретическая механика»
Динамика системы

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

Оценка: выше всех похвал!

Дата: 21.12.24

Москва, 2023

Вариант №8

Задание:

Численно решить дифференциальные уравнения движения механической системы в среде Octave (или Matlab), сделать задание №12 курсовой и построить анимацию движения системы.

Механическая система:

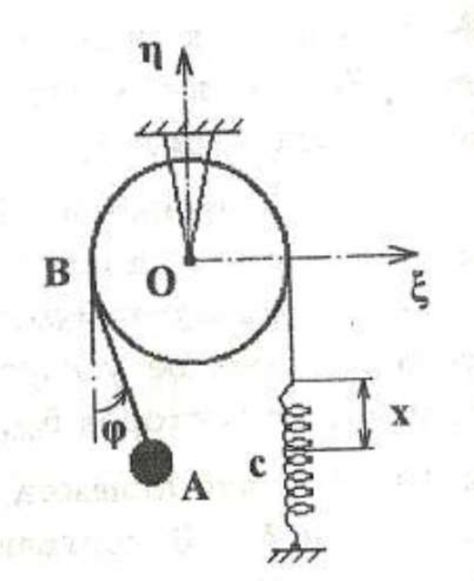

Рис. 8

Уравнения реакции и движения

$$N_\xi = -m\left(\ell\ddot{\varphi} + r\dot{\varphi}^2 + 2\dot{\ell}\dot{\varphi}\right)\cos\varphi - m\left(\ddot{x} - \ell\dot{\varphi}^2\right)\sin\varphi, \quad \ell = \ell_0 + x - r\varphi,$$

$$N_\eta = -m\left(\ell\ddot{\varphi} + r\dot{\varphi}^2 + 2\dot{\ell}\dot{\varphi}\right)\sin\varphi + m\left(\ddot{x} - \ell\dot{\varphi}^2\right)\cos\varphi - cx - (M+m)g.$$

$(\ell \geq 0, \ \ \varphi > -\pi)$

$$[(M/2) + m]\,\ddot{x} - m\ell\dot{\varphi}^2 = mg\cos\varphi - c(x+\delta), \qquad \delta = mg/c,$$

$$\ell\ddot{\varphi} + \dot{\varphi}\,(2\dot{x} - r\dot{\varphi}) = -g\sin\varphi.$$

Текст программы

```
# 0. Импортируем необходимые библиотеки
import numpy as np  # numpy для математических вычислений
import matplotlib.pyplot as plt  # для создания графиков
import matplotlib.pyplot as plot
from matplotlib.animation import FuncAnimation  # для анимации
from scipy.integrate import odeint  # для решения дифференциальных уравнений
```

```python
# 1. Определяем функцию для системы дифференциальных уравнений движения
def EqOfMovement(y, t, M, m, l, r, c, g):
    dy = np.zeros_like(y)
    dy[0] = y[2]  # dx/dt = v
    dy[1] = y[3]  # dφ/dt = ω

    delta = (m * g) / c  # статическое смещение

    # Коэффициенты системы уравнений
    a11 = ((M / 2) + m)
    a12 = 0
    b1 = m * g * np.cos(y[1]) - c * (y[0] + delta) + m * l * y[3] * y[3]

    a21 = 0
    a22 = l
    b2 = -g * np.sin(y[1]) - y[3] * (2 * y[2] - r * y[3])

    # Решение системы уравнений
    dy[2] = (b1 * a22 - b2 * a12)/(a11 * a22 - a21 * a12)  # ускорение по x
    dy[3] = (a11 * b2 - a21 * b1)/(a11 * a22 - a21 * a12)  # угловое
ускорение

    return dy


# 2. Задаем основные параметры системы
STEPS = 750  # количество шагов для расчета
START_VALUE = 0  # начальное время
END_VALUE = 3 * np.pi  # конечное время
M = 1  # масса блока
m = 0.1  # масса груза
r = 0.4  # радиус шкива
l = 1  # длина стержня
c = 50  # жесткость пружины
g = 9.81  # ускорение свободного падения

# 3. Задаем начальные условия
x0 = 0.05  # начальное смещение
phi0 = np.pi / 6  # начальный угол
dx0 = 0  # начальная скорость по x
dphi0 = 0  # начальная угловая скорость
y0 = [x0, phi0, dx0, dphi0]  # вектор начальных условий

# 4. Создаем временной массив и решаем систему уравнений
t = np.linspace(START_VALUE, END_VALUE, STEPS)  # временной массив
Y = odeint(EqOfMovement, y0, t, (M,m,l,r,c,g))  # решение системы
y = Y[:, 0]  # смещение
phi = Y[:, 1]  # угол
dx = Y[:, 2]  # скорость по x
dphi = Y[:, 3]  # угловая скорость

# 5. Вычисляем производные и силы реакции
dY = EqOfMovement(Y, t, M, m, l, r, c, g)
ddx = dY[:, 2]  # ускорение по x
ddphi = dY[:, 3]  # угловое ускорение

l_val = l + y - r * phi  # текущая длина
dl = dx - r * dphi  # производная длины

# Вычисление сил реакции
N_eps = -m * (l_val * ddphi + r * dphi * dphi + 2 * dl * dphi) * np.cos(phi)
- \
```

```
        m * (ddx - l_val * dphi * dphi) * np.sin(phi)
N_nu = -m * (l_val * ddphi + r * dphi * dphi + 2 * dl * dphi) * np.sin(phi) + \
       m * (ddx - l_val * dphi * dphi) * np.cos(phi) - c * y - (M + m) * g

# 6. Создаем фигуру с сеткой для графиков
fgr = plt.figure(figsize=(15, 10))
gs = fgr.add_gridspec(4, 2, width_ratios=[1, 0.8])

# 7. Создаем подграфики
gr = fgr.add_subplot(gs[:, 0])  # область для анимации
gr.axis('equal')

# Создаем 4 графика справа
ax1 = fgr.add_subplot(gs[0, 1])
ax2 = fgr.add_subplot(gs[1, 1])
ax3 = fgr.add_subplot(gs[2, 1])
ax4 = fgr.add_subplot(gs[3, 1])

# 8. Строим графики зависимостей
line_y, = ax1.plot([], [])
ax1.set_title('y(t)')
ax1.grid(True)
ax1.set_xlim(START_VALUE, END_VALUE)
ax1.set_ylim(min(y), max(y))

line_phi, = ax2.plot([], [])
ax2.set_title('phi(t)')
ax2.grid(True)
ax2.set_xlim(START_VALUE, END_VALUE)
ax2.set_ylim(min(phi), max(phi))

line_n_eps, = ax3.plot([], [])
ax3.set_title('N_eps(t)')
ax3.grid(True)
ax3.set_xlim(START_VALUE, END_VALUE)
ax3.set_ylim(min(N_eps), max(N_eps))

line_n_nu, = ax4.plot([], [])
ax4.set_title('N_nu(t)')
ax4.grid(True)
ax4.set_xlim(START_VALUE, END_VALUE)
ax4.set_ylim(min(N_nu), max(N_nu))

# 9. Задаем параметры анимации
XO = 3  # x-координата центра блока
YO = 4  # y-координата центра блока
RB = 0.65  # радиус блока
Y0 = 2.3  # начальная амплитуда
RS = 0.1  # радиус малого круга
NP = 20  # количество витков пружины

# 10. 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

# 11. Расчет координат для движущихся частей
y_l = y + Y0  # левая часть механизма
y_r = y - Y0  # правая часть механизма
Xb = XO - RB  # x-координата точки B
Yb = YO       # y-координата точки B
Xa = Xb + y_l * np.sin(phi)  # x-координата точки A
```

```python
Ya = Yb - y_l * np.cos(phi)  # y-координата точки A

# 12. 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

# 13. 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

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

# 14. 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

# 15. Функция обновления кадров анимации
def run(i):
    # Обновляем положения всех движущихся элементов
    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
    line_y.set_data(t[:i], y[:i])
    line_phi.set_data(t[:i], phi[:i])
    line_n_eps.set_data(t[:i], N_eps[:i])
    line_n_nu.set_data(t[:i], N_nu[:i])

    return [m, AB, Block, Pruzh, line_y, line_phi, line_n_eps, line_n_nu]

# 16. Создаем и показываем анимацию
plt.tight_layout()  # оптимизируем расположение графиков
anim = FuncAnimation(fgr, run, frames=STEPS, interval=1)  # создаем анимацию

# Показываем результат
plt.show()
```

Результат работы:

1) M = 1, m = 0.1, r = 0.4, l = 1, c = 50;

y0 = [0.02 π/6 0 0]