Московский авиационный институт

(национальный исследовательский университет)

Институт № 8 «Информационные технологии и прикладная математика»

**Лабораторная работа №4**

**по курсу «Теоретическая механика»**

**Малые колебания вблизи устойчивого положения равновесия**

Выполнил студент группы М8О-207Б-20

Алапанова Эльза Халилевна

Преподаватель: Чекина Евгения Алексеевна

Оценка:

Дата: 25/12/2021

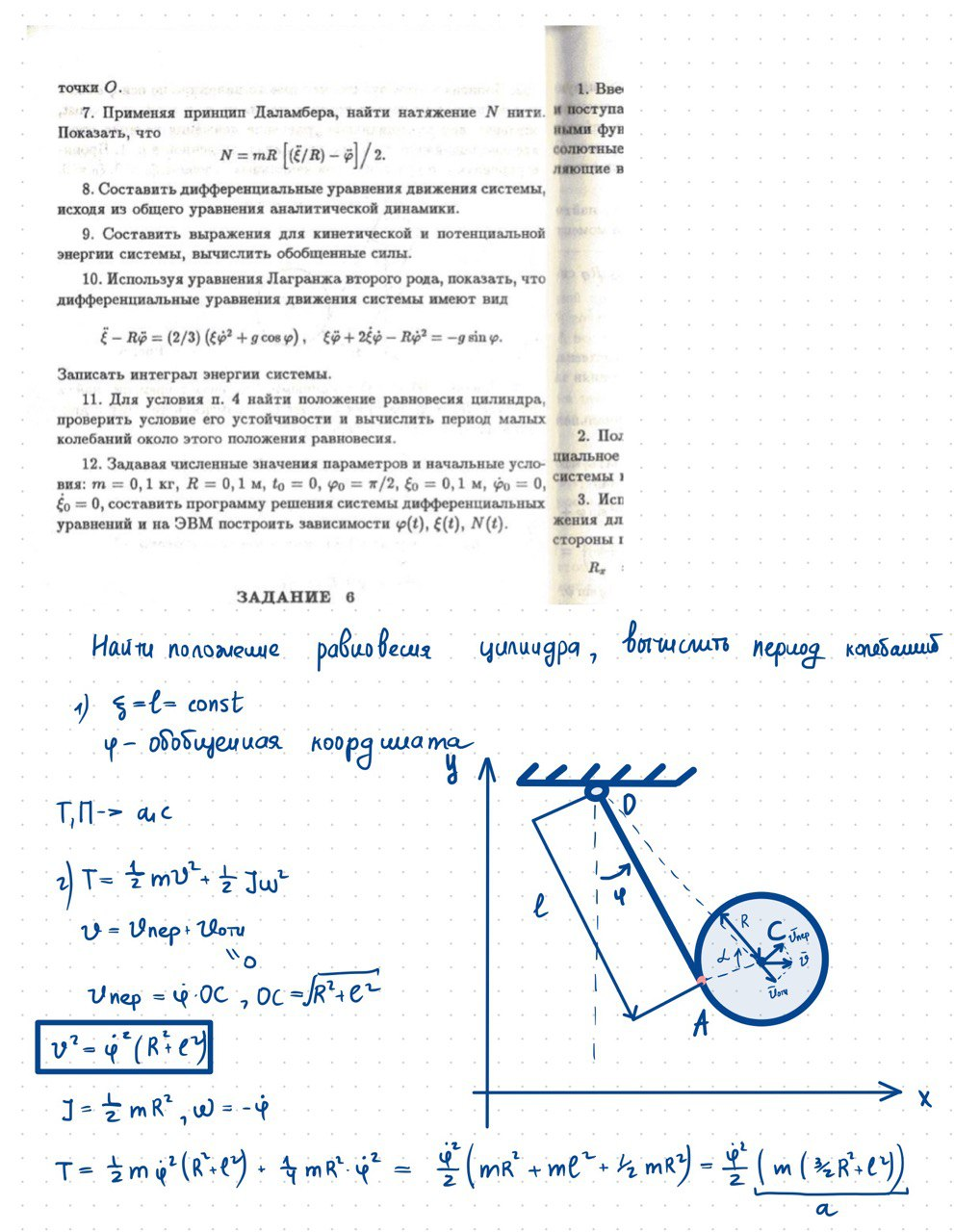
Москва, 2021

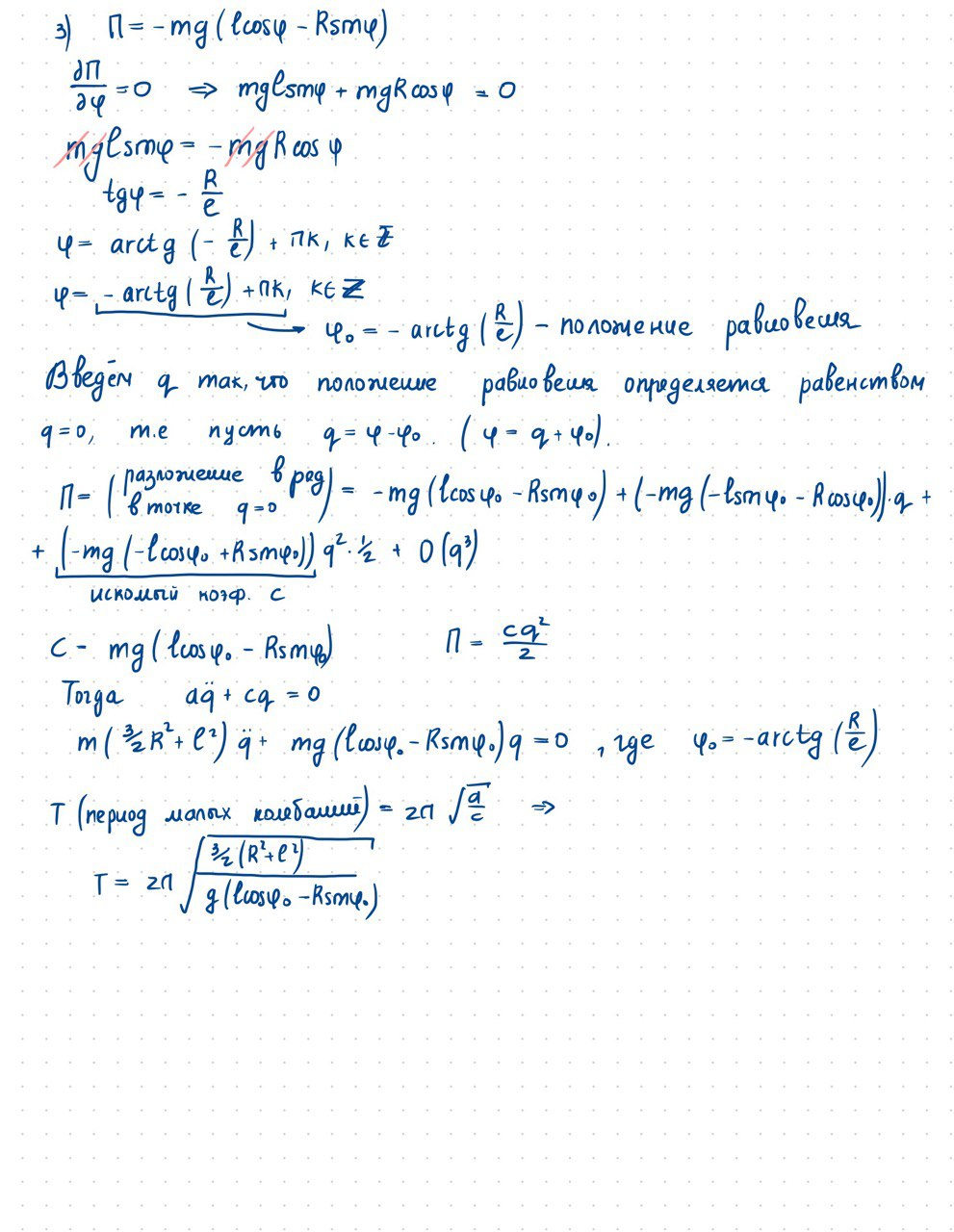
**Вариант №«5»**

**Задание:**

В ограничениях, заданных в пункте 11 методички смоделировать малые колебания исследуемой системы. То есть применить к своей задаче ограничения пункта 11, найти коэффициент **a** для кинетической энергии, коэффициент **с** для потенциальной, составить уравнение Лагранжа.

**Механическая система:**





**Текст программы:**

Основная :

import numpy as np  
import math  
import matplotlib.pyplot as plt  
from matplotlib.animation import FuncAnimation  
import scipy.integrate  
from scipy.integrate import odeint  
import sympy as sp  
  
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  
  
def formY2*(*y, t, fOm*)*:  
 y1,y2 = y  
 dydt = *[*y2,fOm*(*y1,y2*)]* return dydt  
  
#defining parameters  
m = 0.1  
R = 1.5  
phi0 = sp.pi / 2  
ksi0 = 0.1  
g = 9.81  
l = 4  
  
#defining t as a symbol (it will be the independent variable)  
t = sp.Symbol*(*'t'*)*#defining ksi, phi, V=dksi/dt and om=dphi/dt as functions of 't'  
ksi = 0  
phi = sp.Function*(*'phi'*)(*t*)*V = 0  
om = sp.Function*(*'om'*)(*t*)*#constructing the Lagrange equations  
#1 defining the kinetic energy  
v2 = om\*\*2 \* *(*R\*\*2 + ksi\*\*2*)* + V\*\*2 - 2 \* V \* om \* R  
w = V / R - om  
J = 1 / 2 \* m \* R\*\*2  
TT = *(*m\*v2*)*/2 + *(*J\*w\*\*2*)*/2  
# TT = 0.5\*m\*(om\*\*2 \* (R\*\*2+ksi\*\*2) + V\*\*2 - (2 \* om \* V \* R)) + 0.25 \* m \* R\*\*2 \* (V / R - om)\*\*2  
# TT = om / 2 \* (m \* (3/2\*R\*\*2 + l \*\* 2))  
#2 defining potential energy  
Pi = -m\*g\**(*ksi\*sp.cos*(*phi*)*-R\*sp.sin*(*phi*))*#Lagrange function  
L = TT-Pi  
  
#equations  
# ur1 = sp.diff(sp.diff(L,V),t)-sp.diff(L,ksi)  
ur2 = sp.diff*(*sp.diff*(*L,om*)*,t*)*-sp.diff*(*L,phi*)*print*(*ur2*)*#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(ksi,t),V), (sp.diff(phi,t), om)])  
b2 = -ur2.coeff*(*sp.diff*(*om,t*)*,0*)*.subs*(*sp.diff*(*phi,t*)*, om*)*# detA = a11\*a22-a12\*a21  
# detA1 = b1\*a22-b2\*a21  
# detA2 = a11\*b2-b1\*a21  
  
# dVdt = detA1/detA  
domdt = b2/a22  
print*(*domdt*)*countOfFrames = 300  
  
# Constructing the system of differential equations  
T = np.linspace*(*0, 12, countOfFrames*)*# Pay attention here, the function lambdify translate function from the sympy to numpy and then form arrays much more  
# faster then we did using subs in previous lessons!  
# fV = sp.lambdify([ksi,phi,V,om], dVdt, "numpy")  
fOm = sp.lambdify*([*phi,om*]*, domdt, "numpy"*)*y0 = *[*-2, -0.1*]*sol = odeint*(*formY2, y0, T, args = *(*fOm, *))*#sol - our solution  
#sol[:,0] - ksi  
#sol[:,1] - phi  
#sol[:,2] - dksi/dt  
#sol[:,3] - dphi/dt  
  
Ksi = l  
Phi = sol*[*:,0*]*print*(*sol*[*:,0*])*Steps = 300  
t = np.linspace*(*0, 10, Steps*)*X\_O = 3 # координаты точки О  
Y\_O = 10  
  
X\_A = Ksi \* np.sin*(*Phi*)* + X\_O  
Y\_A = - Ksi \* np.cos*(*Phi*)* + Y\_O  
  
X\_C = X\_O + Ksi \* np.sin*(*Phi*)* + R \* np.cos*(*Phi*)*Y\_C = Y\_O - Ksi \* np.cos*(*Phi*)* + R \* np.sin*(*Phi*)*angle = np.linspace*(*0, np.pi\*2, 150*)*X\_Circle = R\*np.cos*(*angle*)*Y\_Circle = R\*np.sin*(*angle*)*X\_Ground = *[*0, 6*]* # это подвес, на котором держится точка О  
Y\_Ground = *[*10, 10*]*lSt= 10  
fig = plt.figure*(*figsize=*[*lSt + 0.5, lSt + 0.5*])*ax = fig.add\_subplot*(*1, 2, 1*)*ax.axis*(*'equal'*)*ax.set*(*xlim=*[*X\_O - *(*lSt + 0.5*)*, X\_O + *(*lSt + 0.5*)]*,  
 ylim=*[*Y\_O/2 - *(*lSt + 0.5*)*, Y\_O/2 + *(*lSt + 0.5*)])*ax.plot*(*X\_Ground, Y\_Ground, color='black', linewidth=3*)*Drawed\_Circle = ax.plot*(*X\_C*[*0*]*+X\_Circle, Y\_C*[*0*]*+Y\_Circle*)[*0*]*Line\_OA = ax.plot*([*X\_O, Y\_O*]*, *[*X\_A*[*0*]*, Y\_A*[*0*]])[*0*]* # линия, соединяющая точки O и А  
  
Point\_O = ax.plot*(*X\_O, Y\_O, marker='o', markersize=10*)[*0*]*Point\_A = ax.plot*(*X\_A*[*0*]*, Y\_A*[*0*]*, marker='o'*)[*0*]*Point\_C = ax.plot*(*X\_C*[*0*]*, Y\_C*[*0*]*, marker='o'*)[*0*]*ax1 = fig.add\_subplot*(*4, 2, 2*)*ax1.plot*(*T, sol*[*:,0*])*plt.xlabel*(*'T'*)*plt.ylabel*(*'Phi'*)*ax1 = fig.add\_subplot*(*4, 2, 6*)*ax1.plot*(*T, sol*[*:,1*])*plt.xlabel*(*'T'*)*plt.ylabel*(*'Om'*)*def Kino*(*i*)*:  
 Point\_O.set\_data*(*X\_O, Y\_O*)* Point\_A.set\_data*(*X\_A*[*i*]*, Y\_A*[*i*])* Line\_OA.set\_data*([*X\_O, X\_A*[*i*]]*, *[*Y\_O, Y\_A*[*i*]])* Point\_C.set\_data*(*X\_C*[*i*]*, Y\_C*[*i*])* Drawed\_Circle.set\_data*(*X\_C*[*i*]*+X\_Circle, Y\_C*[*i*]*+Y\_Circle*)* return *[*Point\_O, Point\_A, Point\_C, Line\_OA*]*anima = FuncAnimation*(*fig, Kino, frames=Steps, interval=10*)*plt.show*()*

**Результат работы:**

