ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕ-НИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ОТЧЕТ

О ВЫПЛОНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №3

«ДИНАМИКА СИСТЕМЫ»

ПО ДИСЦИПЛИНЕ «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА И ОС-НОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ»

ВАРИАНТ ЗАДАНИЯ №15

Выполнил(а) студент группы М8О-208Б-23

Денисов Константин Дмитриевич________________________
подпись, дата

Проверил и принял

Ст. преп. каф. 802 Волков Е.В.______________________________
подпись, дата

с оценкой _______________________________

Москва, 2024

Вариант №15

Задание:

Построить анимацию движения системы, а также графики законов движения системы для разных случаев системы (поэкспериментировать с параметрами системы). Вывести урав-нения, определяющие законы движения системы.

$$(m_1 + m_2)(\ddot{s} - g\sin\alpha) - m_2\ell\left[\ddot{\varphi}\cos(\varphi - \alpha) - \dot{\varphi}^2\sin(\varphi - \alpha)\right] = 0,$$

$$\ell\ddot{\varphi} - \ddot{s}\cos(\varphi - \alpha) + g\sin\varphi = 0.$$

Лабораторная работа №3

Механическая система:

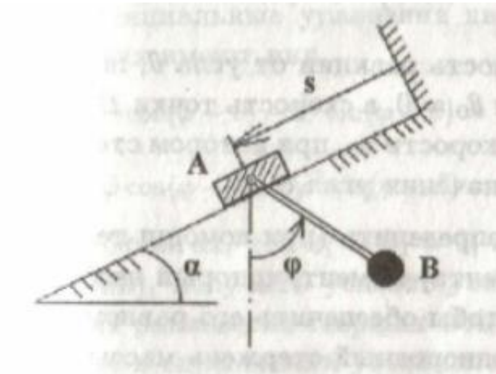


Рис. 15

Для переменных задать следующие значения

12. Задавая численные значения параметров и начальные условия: $m_1 = 1$ кг, $m_2 = 0,5$ кг, $\ell = 0,5$ м, $t_0 = 0$, $s_0 = 0$, $\varphi_0 = \pi/6$, $\dot{s}_0 = 0$, $\dot{\varphi}_0 = 12c^{-1}$, составить программу решения системы дифференциальных уравнений и на ЭВМ построить зависимости $s(t)$, $\varphi(t)$, $N(t)$ для двух значений параметра α: $\alpha = \pi/12$ и $\alpha = \pi/4$.

Текст программы

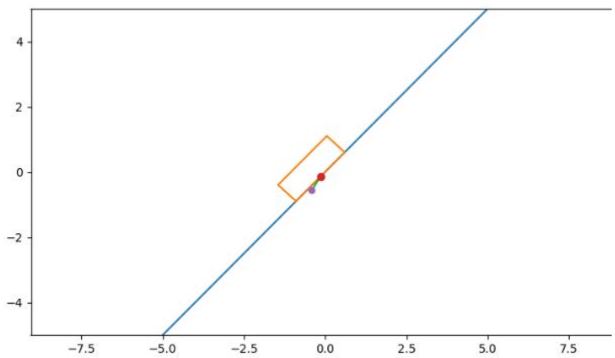
```
import matplotlib  # Импортируем библиотеку для работы с графиками.
import numpy as np  # Импортируем библиотеку для работы с массивами и числовыми данными.
import matplotlib.pyplot as plt  # Импортируем модуль для построения графиков.
from matplotlib.animation import FuncAnimation  # Импортируем класс для создания анимации.
import sympy as sp  # Импортируем библиотеку для символьных вычислений.
import math  # Импортируем библиотеку для работы с математическими функциями.
from scipy.integrate import odeint  # Импортируем функцию для численного решения систем дифференциальных уравнений.
```

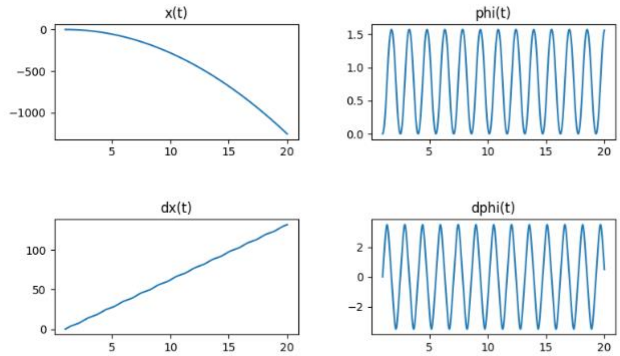
```
matplotlib.use("TkAgg")  # Устанавливаем бэкенд для отображения графиков в отдельном окне.

t = np.linspace(1, 20, 1001)  # Задаём временной интервал от 1 до 20 с 1001 точкой.

# Определяем функцию системы дифференциальных уравнений.
def odesys(y, t, m1, m2, l, g, alpha):
    dy = np.zeros(4)  # Создаём массив для производных.
    dy[0] = y[2]  # Производная x по времени равна скорости dx.
    dy[1] = y[3]  # Производная угла phi равна угловой скорости dphi.

    # Коэффициенты для матричного уравнения.
    a11 = m1 + m2  # Масса тележки и груза.
    a12 = -m2 * l * np.cos(y[1] - alpha)  # Связь тележки и груза.
    a21 = -np.cos(y[1] - alpha)  # Обратная связь.
    a22 = l  # Длина подвеса.

    # Правая часть уравнения.
    b1 = (m1 + m2) * g * np.sin(alpha) - m2 * l * (y[3])**2 * np.sin(y[1] - alpha)  # Горизонтальная сила.
    b2 = -g * np.sin(y[1])  # Вертикальная сила.

    # Решаем систему линейных уравнений для ускорений.
    dy[2] = (b1 * a22 - a12 * b2) / (a11 * a22 - a21 * a12)  # Ускорение dx.
    dy[3] = (b2 * a11 - b1 * a21) / (a11 * a22 - a21 * a12)  # Угловое ускорение dphi.

    return dy  # Возвращаем массив производных.

# Параметры системы.
m1 = 1  # Масса тележки.
m2 = 500  # Масса груза.
l = 5  # Длина подвеса.
g = 9.81  # Ускорение свободного падения.
alpha = math.pi / 4  # Угол наклона системы.

# Начальные условия.
x0 = 0  # Начальное положение тележки.
phi0 = 0  # Начальный угол маятника.
dx0 = 0  # Начальная скорость тележки.
dphi0 = 12  # Начальная угловая скорость маятника.
y0 = [x0, phi0, dx0, dphi0]  # Начальный вектор состояния.

# Решаем систему уравнений.
Y = odeint(odesys, y0, t, (m1, m2, l, g, alpha))  # Численно решаем систему.

# Извлекаем значения из решения.
x = Y[:, 0]  # Положение тележки.
phi = Y[:, 1]  # Угол маятника.
dx = Y[:, 2]  # Скорость тележки.
dphi = Y[:, 3]  # Угловая скорость маятника.

X_0 = 4  # Горизонтальная смещение (не используется в коде).
a = 2.5  # Горизонтальная размерность тележки.
b = 3  # Вертикальная размерность тележки.

# Координаты точек тележки и маятника.
X_A = -a / 40 * x  # Горизонтальная координата тележки.
Y_A = X_A  # Вертикальная координата тележки (аналогично X_A).
Y_B = Y_A - l * np.sin(math.pi / 1.2 - phi)  # Вертикальная координата маятника.
X_B = X_A + l * np.cos(math.pi / 1.2 - phi)  # Горизонтальная координата маятника.

# Контур тележки.
X_Box = np.array([-0.75, -1.3, 0.2, 0.75, -0.75])  # Координаты контуров тележки по x.
```

```
Y_Box = np.array([-0.75, -0.25, 1.25, 0.75, -0.75])  # Координаты контуров тележки по y.

# Прямая линия для оси.
X_Straight = [-10, 0, 10]  # Координаты линии по x.
Y_Straight = [-10, 0, 10]  # Координаты линии по y.

# Создаём окно для визуализации движения.
fig = plt.figure(figsize=[9, 5])  # Создаём окно с заданным размером.
ax = fig.add_subplot(1, 1, 1)  # Добавляем оси.
ax.axis('equal')  # Устанавливаем одинаковый масштаб по осям.
ax.set(xlim=[-5, 5], ylim=[-5, 5])  # Устанавливаем пределы отображения.

ax.plot(X_Straight, Y_Straight)  # Рисуем прямую ось.
Drawed_Box = ax.plot(X_A[0] + X_Box, Y_A[0] + Y_Box)[0]  # Рисуем тележку.
Line_AB = ax.plot([X_A[0], X_B[0]], [Y_A[0], Y_B[0]])[0]  # Рисуем маятник как линию.
Point_A = ax.plot(X_A[0], Y_A[0], marker='o')[0]  # Рисуем точку на тележке.
Point_B = ax.plot(X_B[0], Y_B[0], marker='o', markersize=5)[0]  # Рисуем точку на маятнике.

# Создаём второе окно для графиков зависимостей.
fig2 = plt.figure(figsize=[9, 5])  # Создаём окно.
ax2 = fig2.add_subplot(2, 2, 1)  # Первый график.
ax2.plot(t, -x)  # График положения тележки.
plt.title('x(t)')  # Заголовок графика.

ax3 = fig2.add_subplot(2, 2, 2)  # Второй график.
ax3.plot(t, phi)  # График угла маятника.
plt.title('phi(t)')  # Заголовок графика.

ax4 = fig2.add_subplot(2, 2, 3)  # Третий график.
ax4.plot(t, dx)  # График скорости тележки.
plt.title('dx(t)')  # Заголовок графика.

ax5 = fig2.add_subplot(2, 2, 4)  # Четвёртый график.
ax5.plot(t, dphi)  # График угловой скорости маятника.
plt.title('dphi(t)')  # Заголовок графика.

plt.subplots_adjust(wspace=0.3, hspace=0.7)  # Устанавливаем расстояние между графиками.

# Функция для анимации движения.
def Kino(i):
    Point_A.set_data(X_A[i], Y_A[i])  # Обновляем положение точки тележки.
    Point_B.set_data(X_B[i], Y_B[i])  # Обновляем положение точки маятника.
    Line_AB.set_data([X_A[i], X_B[i]], [Y_A[i], Y_B[i]])  # Обновляем линию маятника.
    Drawed_Box.set_data(X_A[i] + X_Box, Y_A[i] + Y_Box)  # Обновляем положение тележки.
    return [Point_A, Point_B, Line_AB, Drawed_Box]  # Возвращаем обновлённые элементы.

# Создаём анимацию.
anima = FuncAnimation(fig, Kino, frames=1000, interval=50)  # Анимация с 1000 кадрами и интервалом 50
мс.
plt.show()  # Показываем графики и анимацию.
```

Работа программы:

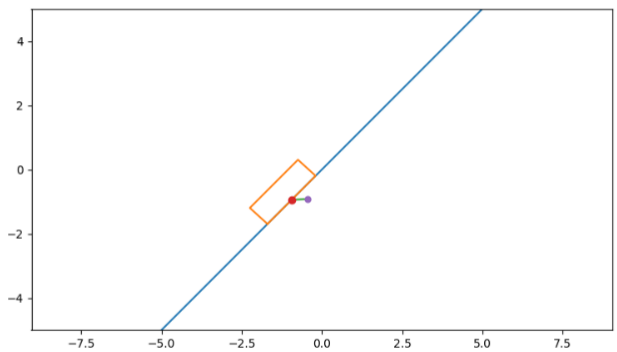
При начальных параметрах: m1 = 1; m2 = 0.5; l = 0.5; x0 = 0; phi0 = pi/12; dx0 = 0; dphi0 = 12

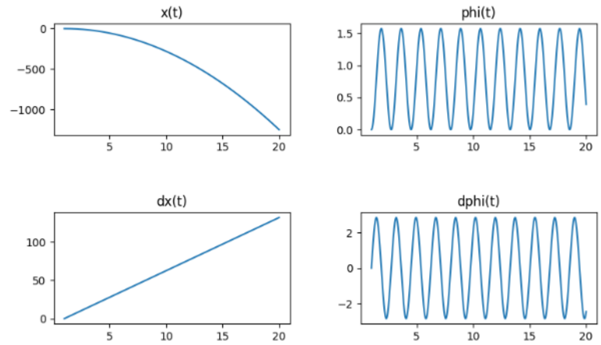




Результат : Тело движется вниз с нормальной скоростью, ускоряясь.

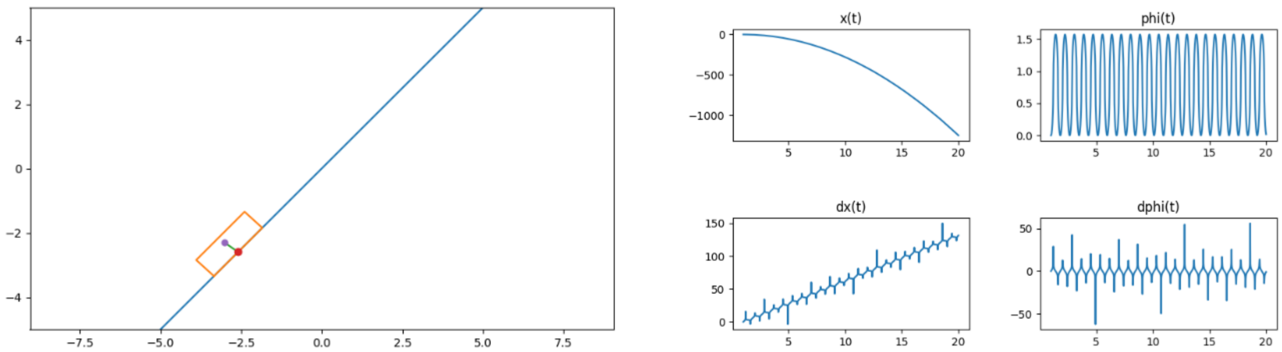
2) При параметрах: m1 = 1000; m2 = 0.5; l = 0.5; x0 = 0; phi0 = pi/12; dx0 = 0; dphi0 = 12





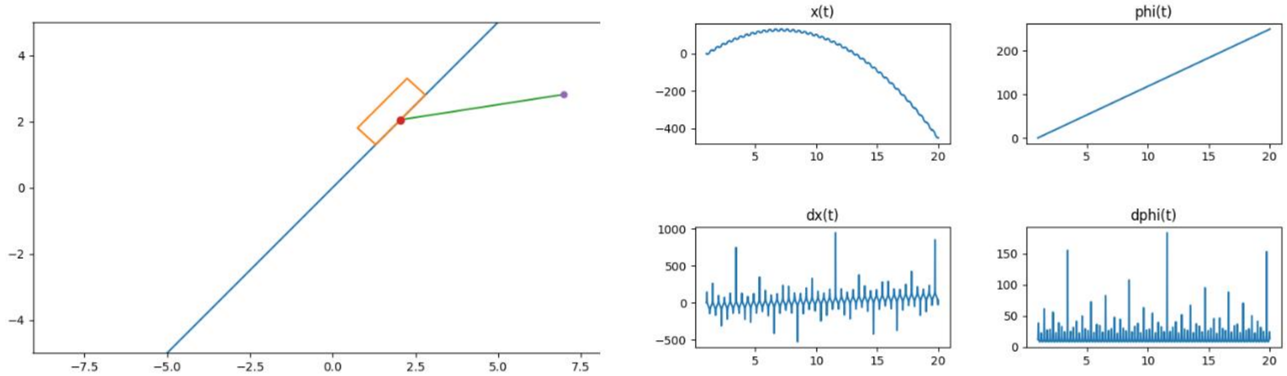
Результат: Тело стало двигаться значительно медленней, маятник колеблется в том же режиме.

3) При параметрах: m1 = 1; m2 = 500; l = 0.5; x0 = 0; phi0 = pi/12; dx0 = 0; dphi0 = 12



Результат: В интервале от pi/2 до -pi/2 скорость движу маятника возросла, также маятник стал замирать в этих точках, скорость движения тела практически увеличилась.

4) При параметрах: m1 = 1; m2 = 500; l = 5; x0 = 0; phi0 = 0; dx0 = 0; dphi0 = 12



Результат: Из-за увеличения длины нити тело стало двигаться в противоположном на-правлении.

Вывод

ходе лабораторной работы я запрограммировал уравнение Лагранжа в анимацию 2ой лабораторной работы, отладил программу, поэкспериментировали с начальными значени-ями и получил несколько случаев поведения системы, которые были рассмотрены с при-ведением графиков и скриншотов анимации.