Лабораторные работы по теоретической механике

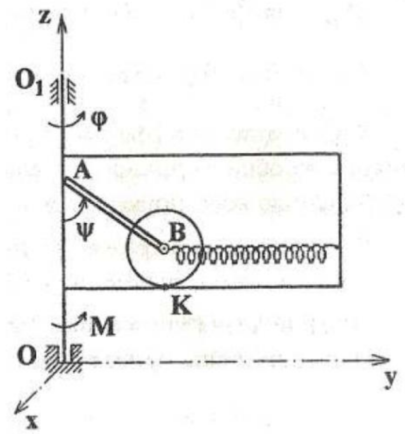
Моделирование динамических систем средствами языка Python

Интегрирование системы

Нам предстоит просимулировать движение системы. Вначале требуется составить уравнения движения системы, и затем их решить. Решение полученных уравнений будет численным, с помощью компьютера.

Уравнения движения механических систем можно получить с помощью основных теорем динамики или с помощью уравнений Лагранжа. Так как в наших системах имеется множество связей (шарниров, опор, точек контакта) и, следовательно, реакций, лучше будет записать именно уравнения Лагранжа.

Для следующей системы, приведённой на рисунке, уравнения движения имеют вид:



$$\left\{J + \left(\frac{m}{3} + m_1\right) l^2 \sin^2\psi + \frac{m_1}{4} r^2\right\} \ddot{\varphi} + \left(\frac{m}{3} + m_1\right) l^2 \dot{\varphi}\dot{\psi} \sin 2\psi = -\alpha\varphi,$$

$$\left(\frac{m}{3} + \frac{3}{2} m_1 \cos^2\psi\right) \ddot{\psi} - \frac{1}{2}\left\{\left(\frac{m}{3} + m_1\right) \dot{\varphi}^2 + \frac{3}{2} m_1 \dot{\psi}^2\right\} \sin 2\psi = \frac{1}{2}\left(\frac{mg}{l} \sin\psi - c \sin 2\psi\right).$$

Здесь J – момент инерции рамки относительно оси Oz, m и l – масса и длина однородного стержня AB, m_1 и r – масса и радиус однородного диска, c – жёсткость пружины (длина которой в недеформированном состоянии равна a – длине рамки) и α – константа, характеризующая момент $M = -\alpha\varphi$, приложенный к рамке.

Численное интегрирование полученных уравнений производится функцией odeint из библиотеки scipy.integrate. Импортируем необходимые библиотеки и функции

```
import numpy as n
from scipy.integrate import odeint
import matplotlib.pyplot as plt
```

Для работы функции численного интегрирования на основе полученных уравнений требуется составить функцию вида $\dot{y} = f(y, t)$, принимающую на вход вектор состояния системы y и время t. В нашем случае система описывается двумя уравнениями второго порядка, следовательно, за вектор состояния системы можно принять $y = (y_1, y_2, y_3, y_4) = \left(\varphi, \psi, \dot{\varphi}, \dot{\psi}\right)$. Тогда вектор производных имеет вид $\dot{y} = (\dot{y}_1, \dot{y}_2, \dot{y}_3, \dot{y}_4) = \left(\dot{\varphi}, \dot{\psi}, \ddot{\varphi}, \ddot{\psi}\right)$. Для установленного формата функции потребуется уравнения движения разрешить относительно вторых производных. Это можно сделать, например, с помощью правила Крамера прямо в программе, как показано ниже.

Запишем в начале программы функцию, описывающую уравнения движения (назовём её SystDiffEq):

```
def SystDiffEq(y,t,J,m,m1,r,l,alph,g,c):
    # y[0,1,2,3] = phi,psi,phi',psi'
    # dy[0,1,2,3] = phi',psi',phi'',psi''
    dy = np.zeros_like(y)
    dy[0] = y[2] #тривиальные уравнения вида
    dy[1] = y[3] #dphi = phi’, dpsi = psi’
    # представим систему уравнений движения в виде
    # линейной относительно вторых производных
    # системы  A*q'' = B, где
    #   q = (phi; psi), A = A(phi, psi),
    #   B = B(phi, psi, phi', psi'):
    #
    # a11 * phi'' + a12 * psi'' = b1
    # a21 * phi'' + a22 * psi'' = b2
    # коэффициенты первого уравнения
```

```
        a11 = (J + (m/3 + m1) * l**2 * n.sin(y[1])**2 + m1/4 * r**2)
        a12 = 0
        b1 = - (m/3 + m1) * l**2 * y[2] * y[3] * n.sin(2*y[1]) - alph * y[0]
        # коэффициенты второго уравнения
        a21 = 0
        a22 = (m/3 + 3/2 * m1 * n.cos(y[1])**2)
        b2 = 1/2 * ((m/3 + m1) * y[2]**2 + 3/2 * m1 * y[3]**2) * n.sin(2*y[1]) + 1/2 * (m*g/l * n.sin(y[1]) - c * n.sin(2*y[1]))
        # решение правилом Крамера
        dy[2] = (b1*a22 - b2*a12)/(a11*a22 - a12*a21)
        dy[3] = (a11*b2 - b1*a21)/(a11*a22 - a12*a21)
        return dy
```

Отметим, что после аргументов y и t идут иные параметры задачи – массы, длины, жёсткости и прочие.

Теперь приготовим всё необходимое для численного интегрирования: зададим параметры задачи, начальное состояние системы и сетку моментов времени, в которые требуется выдать результат

```
# Определяем величины, заданные для системы
J = 20
m = 2
m1 = 1
l = 1
r = 0.2
alph = 30
c = 20
g = 9.81
# Определяем начальные условия
phi0 = n.pi/6
```

```
psi0 = n.pi/12
dphi0 = 0.1
dpsi0 = 0
y0 = [phi0, psi0, d phi0, dpsi0]
# Определяем сетку времени
T = n.linspace(0, 10, 100)
```

Вызовем функцию численного интегрирования уравнений движения

```
Y = odeint(SystDiffEq, y0, T, (J, m, m1, r, l, alph, g, c))
```

Здесь первый аргумент – описанная выше функция системы уравнений движения, $y0$ – вектор начального состояния системы, T – сетка по времени, в моменты которой требуется выдать результат, далее в скобках идёт набор дополнительных аргументов функции уравнений SystDiffEq. Переменная Y после работы функции – это матрица, строки которой представляют собой значения вектора $y = (\varphi, \psi, \dot{\varphi}, \dot{\psi})$ в моменты времени T.

Считаем результат из полученной матрицы Y

```
Phi = Y[:,0]
Psi = Y[:,1]
Phit = Y[:,2]
Psit = Y[:,3]
```

и построим графики полученных решений

```
fgr = plt.figure(figsize=[7,11])
pltPhi = fgr.add_subplot(4,1,1)
pltPhi.plot(T, Phi)
pltPhi.set_title('Phi(t)')
pltPsi = fgr.add_subplot(4,1,2)
pltPsi.plot(T, Psi)
pltPsi.set_title('Psi(t)')
```

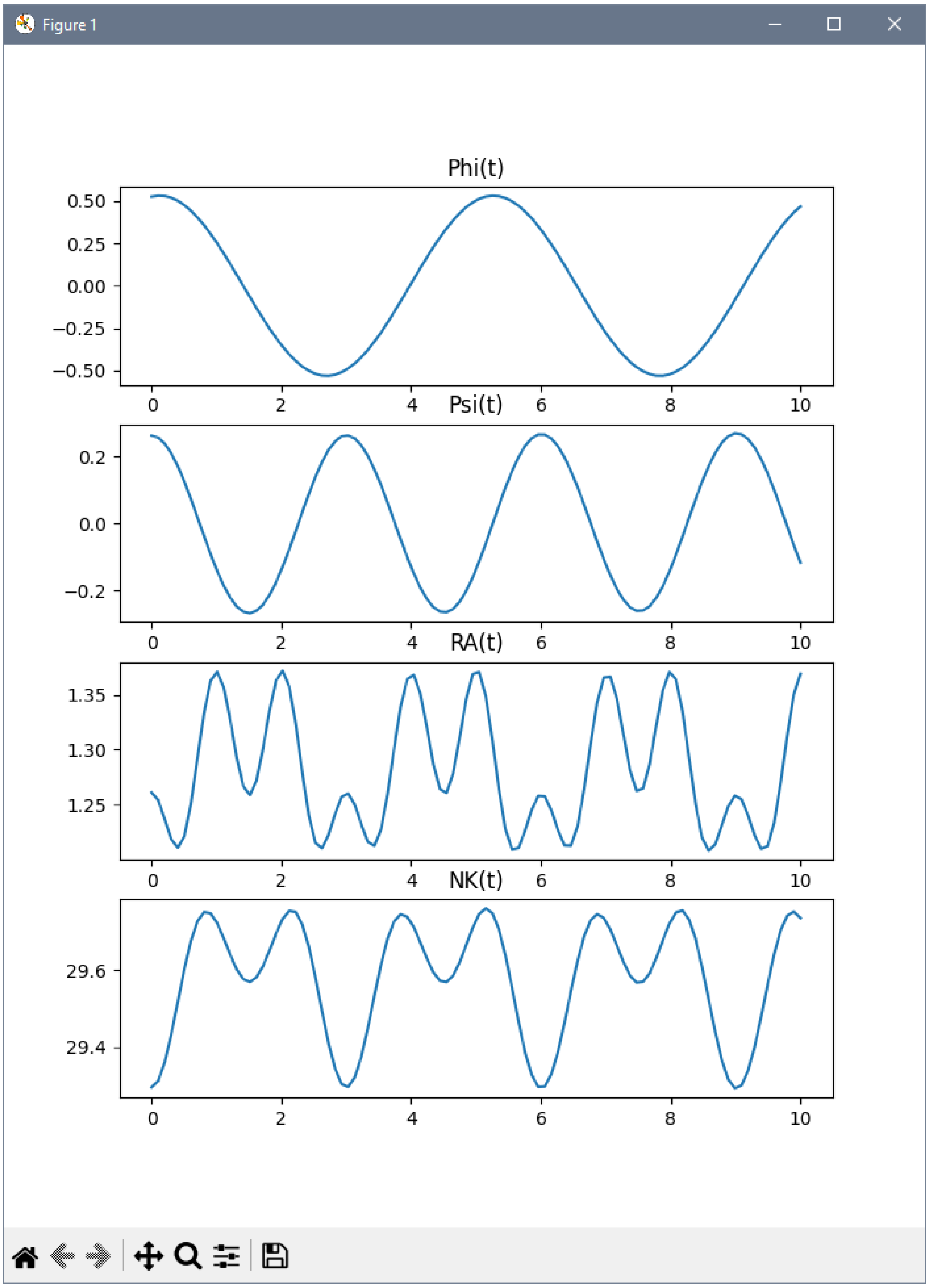
Также добавим графики реакций, указанных в задании. Для этого нам потребуется вычислить вторые производные координат по времени, в чём нам поможет ранее написанная функция SystDiffEq:

```
Phitt = n.zeros_like(T)

Psitt = n.zeros_like(T)

for i in range(len(T)):

    Phitt = SystDiffEq([Phi[i], Psi[i], Phit[i], Psit[i]], T[i], J,m,m1,r,l,alph,g,c)[2]

    Psitt = SystDiffEq([Phi[i], Psi[i], Phit[i], Psit[i]], T[i], J,m,m1,r,l,alph,g,c)[3]
```

Выписываем функции реакций и строим их графики:

```
RA = (m + 3*m1)*l * (Psitt * n.cos(Psi) - Psit**2 * n.sin(Psi))/2

pltRA = fgr.add_subplot(4,1,3)

pltRA.plot(T, RA)

pltRA.set_title('RA(t)')


NK = -m/2 * l * (Psitt * n.sin(Psi) - Psit**2 * n.cos(Psi)) + (m+m1)*g

pltNK = fgr.add_subplot(4,1,4)

pltNK.plot(T, NK)

pltNK.set_title('NK(t)')
```

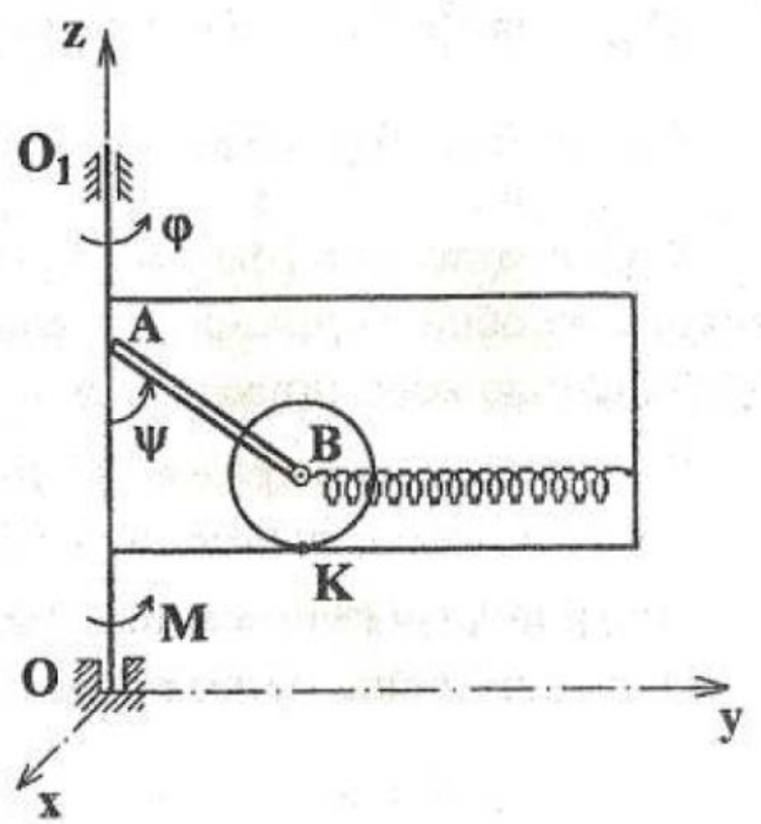
Результат работы программы:



<u>Задание:</u> численно проинтегрируйте уравнения движения системы и постройте графики зависимости от времени координат и указанных реакций.

Анимация системы

Создадим теперь анимацию движения системы. В рамках курсовой работы вы проводите исследование системы с двумя степенями свободы, и в этой лабораторной работе мы построим анимацию её движения. Построим анимацию движения следующей системы в плоскости Ozy.



Здесь диск с центром в точке B двигается по горизонтальному краю прямоугольной рамки, и к нему приделан стержень AB, отклоняющийся на угол $\psi(t)$ от вертикальной оси OO_1. Также в системе имеется обычная пружина. Для начала импортируем необходимую для анимации функцию

```
from matplotlib.animation import FuncAnimation
```

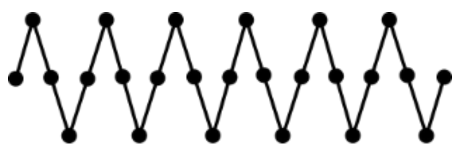
Дальнейший код программы является продолжением программы, где выполнена интегрирование системы дифференциальных уравнений:

```
# Создаём окно для анимации:

fgranim = plt.figure()

an = fgranim.add_subplot(1,1,1)

an.axis('equal')
```

```
# Зададим некоторые геометрические характеристики
h = 0.5 # - высота от точки O до нижнего края рамки
a = 1.5 # - длина рамки
b = 1.5 # - высота рамки

an.plot([0, 0],[0, 3]) # - рисую ось OO_1
an.plot([0, a, a, 0],[h, h, h+b, h+b]) # - рисую рамку

# Шаблон окружности с центром в точке с координатами (0,0)
bet = n.linspace(0, 2*n.pi,100)
Xd = r*n.cos(bet)
Yd = r*n.sin(bet)

# Вычисляем центр диска
Xc = l*n.sin(Psi[0])
Yc = h + r

Disk = an.plot(Xd + Xc, Yd + Yc)[0] # - рисую диск
# Рисую стержень AB:
AB = an.plot([0, Xc],[h + r + l*n.cos(Psi[0]), Yc])[0]
```

Построим теперь шаблон обычной пружины. Зададим количество её витков, ширину, и зададим координаты горизонтального шаблона пружины. Игриковые координаты будут менять свой знак, а иксовые координаты равномерно распределим от нуля до единицы. В дальнейшем иксовые координаты будем домножать на требуемую длину и, в случае необходимости, домножать полученные массивы на матрицу поворота.

```
Np = 20
Xp = n.linspace(0,1,2*Np+1)
```



```
Yp = n.zeros(2*Np+1)
ss = 0
for i in range(2*Np+1):
    Yp[i] = 0.05 * n.sin(ss) # 0.05 - "ширина" пружины
    ss += n.pi/2
```

Построим шаблон спиральной пружины (этот шаблон добавим в качестве демонстрации; в системе свяжем её диск и стержень). Зададим число витков спиральной пружины, наименьший и наибольший радиусы, создадим массив углов промежуточных точек и посчитаем координаты этих точек.

```
Ns = 2
r1 = 0.03
r2 = 0.1
differ = n.linspace(0,1,50*Ns+1)
Betas = differ*(Ns*2*n.pi + Psi[0])
Xs = (r1 + (r2 - r1)*differ)*n.cos(Betas + n.pi/2)
Ys = (r1 + (r2 - r1)*differ)*n.sin(Betas + n.pi/2)
# Рисую пружины:
Pruzh = an.plot(a + (Xc - a)*Xp, h+r + Yp)[0]
SpPruzh = an.plot(Xs + Xc, Ys + Yc)[0]
```

Запишем функцию изменения кадров:

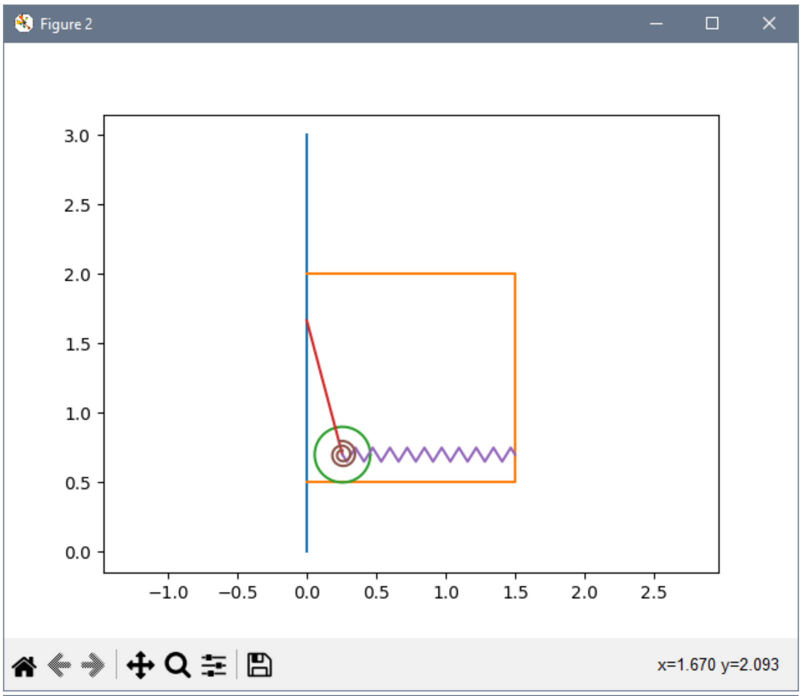
```
def run(i):
    Xc = l*n.sin(Psi[i])
    Disk.set_data(Xd + Xc, Yd + Yc)
    AB.set_data([0, Xc],[h + r + l*n.cos(Psi[i]), Yc])
    Pruzh.set_data(a + (Xc - a)*Xp, h+r + Yp)

    Betas = differ*(Ns*2*n.pi + Psi[i])
    Xs = (r1 + (r2 - r1)*differ)*n.cos(Betas + n.pi/2)
```

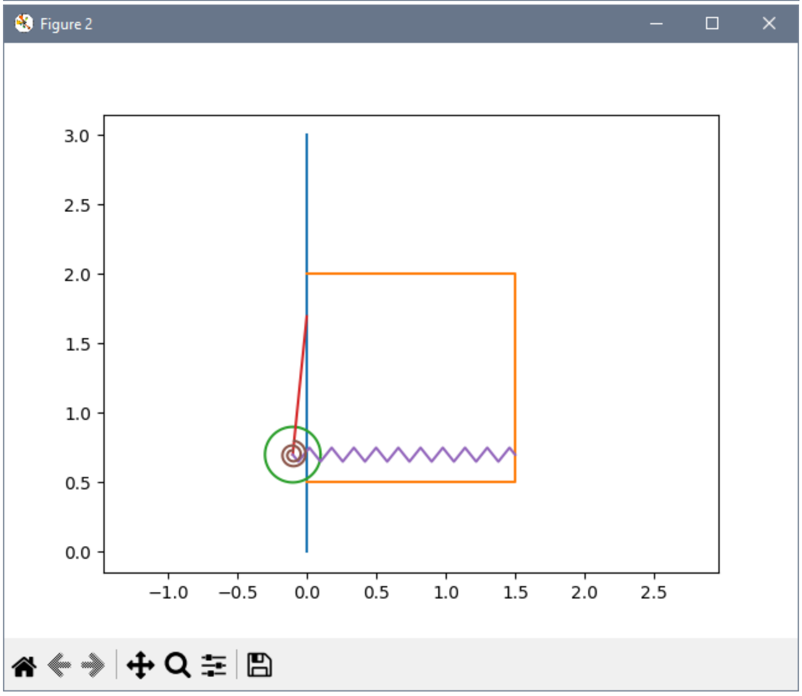
```
Ys = (r1 + (r2 - r1)*differ)*n.sin(Betas + n.pi/2)
SpPruzh.set_data(Xs + Xc, Ys + Yc)
```

Теперь запустим анимацию

```
anim = FuncAnimation(fgranim, run, frames = len(T), interval = 100)
plt.show()
```

Результат работы программы:





Замечание: «выход за пределы» системы, который видно на втором рисунке – нормально при интегрировании системы, так как данная связь не была учтена в уравнениях движения.

Таким образом, мы построили визуализацию движения механической системы с двумя степенями свободы.

Задание: для своего варианта постройте анимацию движения системы с двумя степенями свободы.