ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

**ОТЧЕТ**

**О ВЫПЛОНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

**«ДИНАМИКА СИСТЕМЫ»**

**ПО ДИСЦИПЛИНЕ «ТЕОРЕТИЧЕСКАЯ МЕХАНИКА И ОСНОВЫ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ»**

**ВАРИАНТ ЗАДАНИЯ № 2**

Выполнил(а) студент группы М8О-208Б-21

Белоносов К.А.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Проверил и принял

Зав. каф. 802, Бардин Б.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

*Задание:* проинтегрировать систему дифференциальных уравнений движения системы с двумя степенями свободы с помощью средств Python. Построить анимацию движения системы, а также графики законов движения системы и указанных в задании реакций для разных случаев системы. Исследовать на устойчивость.

*Схема программы:*

Для решения поставленных задач требуется сделать следующие шаги.

Отдельно от основной программы с помощью уравнений движения системы требуется сформировать функцию, которая будет принимать в себя значения , а на выход вернёт значения .

В основной программе требуется задать значения всех параметров, начальное положение системы, и запустить процедуру численного интегрирования системы.

Результаты численного интегрирования системы далее следует использовать при построении анимации движения системы.

*Составление функции уравнений движения:*

Запишем функцию f, которая будет принимать в качестве аргументов время t (даже в случае автономных уравнений этот аргумент нельзя опускать) и вектор состояния системы , а на выход вернёт .

Рассмотрим механическую систему, изображённую на рисунке. Уравнения её движения имеют вид:

Вектор y имеет вид . Система уравнений движения является линейной относительно величин , и её можно решить, правилом Крамера.

Функция имеет вид:

def odesys(y, t, m1, m2, c, R, l, g):

dy = np.zeros(4)

dy[0] = y[2]

dy[1] = y[3]

e = R / (3 \*\* 0.5)

a = y[0] - y[1] - (math.pi / 6)

a11 = (5/6 \* m1 + 4/3 \* m2) \* (R \*\* 2)

a12 = 2 \* m2 \* l \* e \* np.sin(a)

a21 = 2 \* e \* np.sin(a)

a22 = l

b1 = (y[3] \*\* 2) \* np.cos(a) \* 2 \* m2 \* l \* e + \

(m1 \* np.sin(y[0]) + 2 \* m2 \* np.cos(y[0] - math.pi / 6)) \* e \* g - c \* y[0]

b2 = -2 \* e \* (y[2] \*\* 2) \* np.cos(a) - g \* np.sin(y[1])

dy[2] = (b1\*a22 - b2\*a12)/(a11\*a22 - a12\*a21)

dy[3] = (b2\*a11 - b1\*a21)/(a11\*a22 - a12\*a21)

return dy

*Численное интегрирование системы уравнений:*

Численное интегрирование системы дифференциальных уравнений в указанной выше форме производится с помощью функции odeint.

В начале основной программы запишем:

Y = odeint(odesys, y0, t, (m1, m2, c, R, l, g))

phi = Y[:, 0]

ksi = Y[:, 1]

dphi = Y[:, 2]

dksi = Y[:, 3]

ddphi = [odesys(y, t, m1, m2, c, R, l, g)[2] for y, t in zip(Y, t)]

ddksi = [odesys(y, t, m1, m2, c, R, l, g)[3] for y, t in zip(Y, t)]

*Построение графиков:*

Построим графики решения и реакции. Для построения графика реакции потребуется дополнительно вычислить значения . Это можно сделать, вызвав функцию с уравнениями движения, отправив её в качестве аргументов полученное решение.

fig\_for\_graphs = plt.figure(figsize=[13, 7])

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 1)

ax\_for\_graphs.plot(t, phi, color='blue')

ax\_for\_graphs.set\_title("phi(t)")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 2)

ax\_for\_graphs.plot(t, ksi, color='red')

ax\_for\_graphs.set\_title('ksi(t)')

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 3)

ax\_for\_graphs.plot(t, dphi, color='green')

ax\_for\_graphs.set\_title("phi'(t)")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 4)

ax\_for\_graphs.plot(t, dksi, color='black')

ax\_for\_graphs.set\_title('ksi\'(t)')

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 5)

ax\_for\_graphs.plot(t, ddphi, color='green')

ax\_for\_graphs.set\_title("phi''(t)")

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

ax\_for\_graphs = fig\_for\_graphs.add\_subplot(2, 3, 6)

ax\_for\_graphs.plot(t, ddksi, color='black')

ax\_for\_graphs.set\_title('ksi\'\'(t)')

ax\_for\_graphs.set(xlim=[0, t\_fin])

ax\_for\_graphs.grid(True)

*Построение анимации:*

Теперь построим анимацию движения системы, используя полученные результаты. Оформим графическое окно, создадим нарисованные объекты в начальном положении и запустим цикл, переставляющий объекты в положения, отвечающие новым моментам времени.

psi = np.linspace(0, 2 \* math.pi, 100)

C = 1

R1 = 0.2

R2 = R/(3\*\*0.5)/2

X\_Wheel = R\*np.sin(psi)

Y\_Wheel = R\*np.cos(psi)

X\_C = (R/(3\*\*0.5))\*np.sin(phi)

Y\_C = (R/(3\*\*0.5))\*np.cos(phi)

X\_A = X\_C + R\*np.sin(phi - math.pi / 2)

Y\_A = Y\_C + R\*np.cos(phi - math.pi / 2)

X\_B = X\_A + l\*np.cos(ksi)

Y\_B = Y\_A + l\*np.sin(ksi)

thetta = np.linspace(0, C\*math.pi\*2-phi[0], 100)

X\_SpiralSpr = -(R1 + thetta\*(R2-R1)/thetta[-1])\*np.sin(thetta)

Y\_SpiralSpr = (R1 + thetta\*(R2-R1)/thetta[-1])\*np.cos(thetta)

fig = plt.figure(figsize=[15, 7])

ax = fig.add\_subplot(1, 1, 1)

ax.axis('equal')

ax.set(xlim=[-6, 6], ylim=[-6, 6])

Circ = ax.plot(X\_C[0]+R \* np.cos(psi), Y\_C[0]+R \* np.sin(psi), 'yellow')[0]

Point\_O = ax.plot(0, 0, marker='o')[0]

Point\_C = ax.plot(X\_C[0], Y\_C[0], marker='o')[0]

Point\_A = ax.plot(X\_A[0], Y\_A[0], marker='o')[0]

Point\_B = ax.plot(X\_B[0], Y\_B[0], marker='o', markersize=20)[0]

Line\_CO = ax.plot([0, X\_C[0]], [0, Y\_C[0]], 'blue')[0]

Line\_AB = ax.plot([X\_A[0], X\_B[0]], [Y\_A[0], Y\_B[0]])[0]

Drawed\_SpiralSpring = ax.plot(X\_SpiralSpr, Y\_SpiralSpr)[0]

def anima(i):

Point\_C.set\_data(X\_C[i], Y\_C[i])

Point\_A.set\_data(X\_A[i], Y\_A[i])

Point\_B.set\_data(X\_B[i], Y\_B[i])

Line\_AB.set\_data([X\_A[i], X\_B[i]], [Y\_A[i], Y\_B[i]])

Circ.set\_data(X\_C[i] + R \* np.cos(psi), Y\_C[i] + R \* np.sin(psi))

Line\_CO.set\_data([0, X\_C[i]], [0, Y\_C[i]])

thetta = np.linspace(0, C \* math.pi \* 2 - phi[i], 100)

X\_SpiralSpr = -(R1 + thetta \* (R2 - R1) / thetta[-1]) \* np.sin(thetta)

Y\_SpiralSpr = (R1 + thetta \* (R2 - R1) / thetta[-1]) \* np.cos(thetta)

Drawed\_SpiralSpring.set\_data(X\_SpiralSpr, Y\_SpiralSpr)

return [Point\_C, Point\_A, Point\_B, Circ, Line\_CO, Line\_AB, Drawed\_SpiralSpring]

anim = FuncAnimation(fig, anima, frames=len(t), interval=3)

plt.show()

*Примеры работы программы:*

1. Начальные условия:

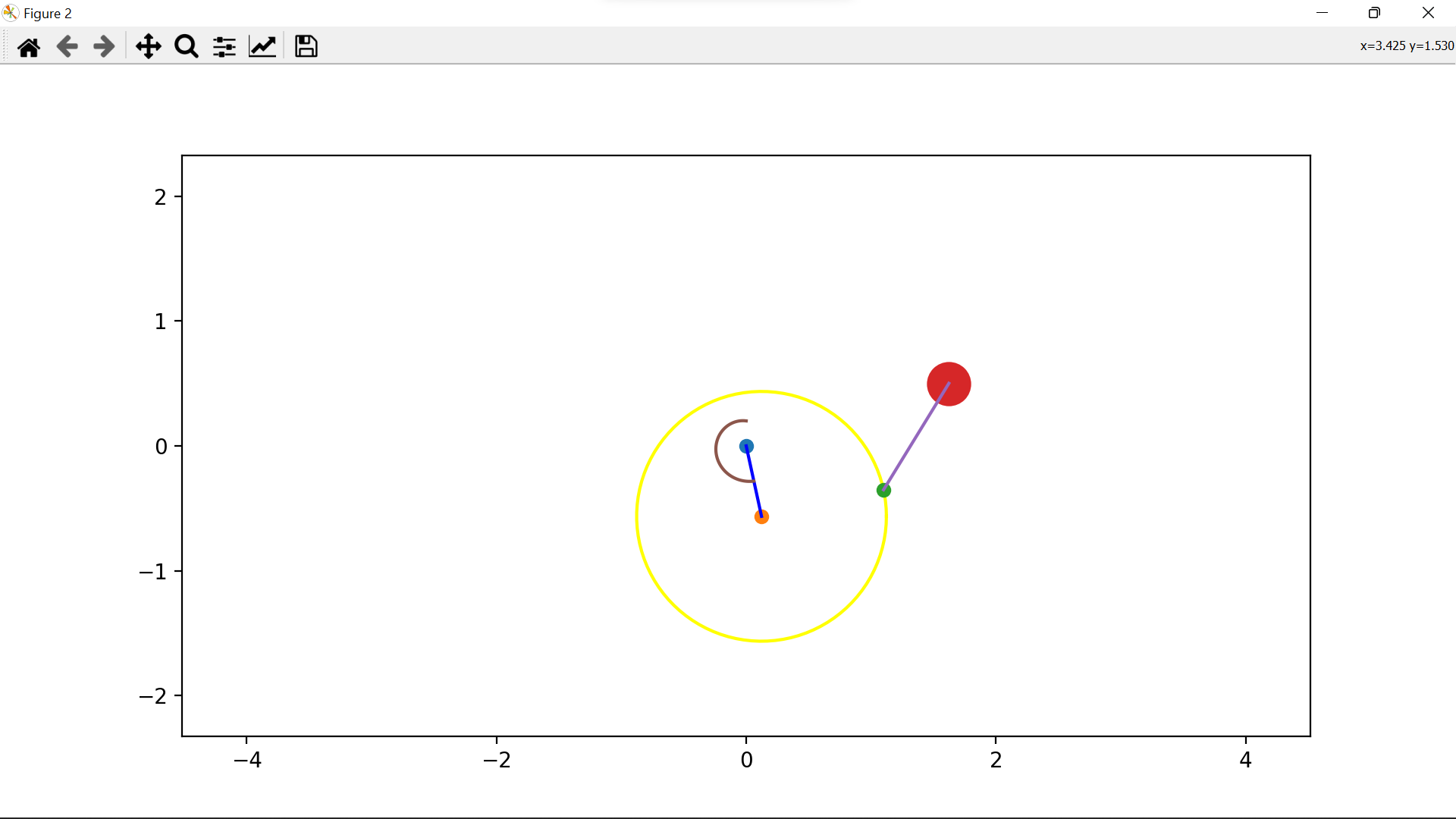
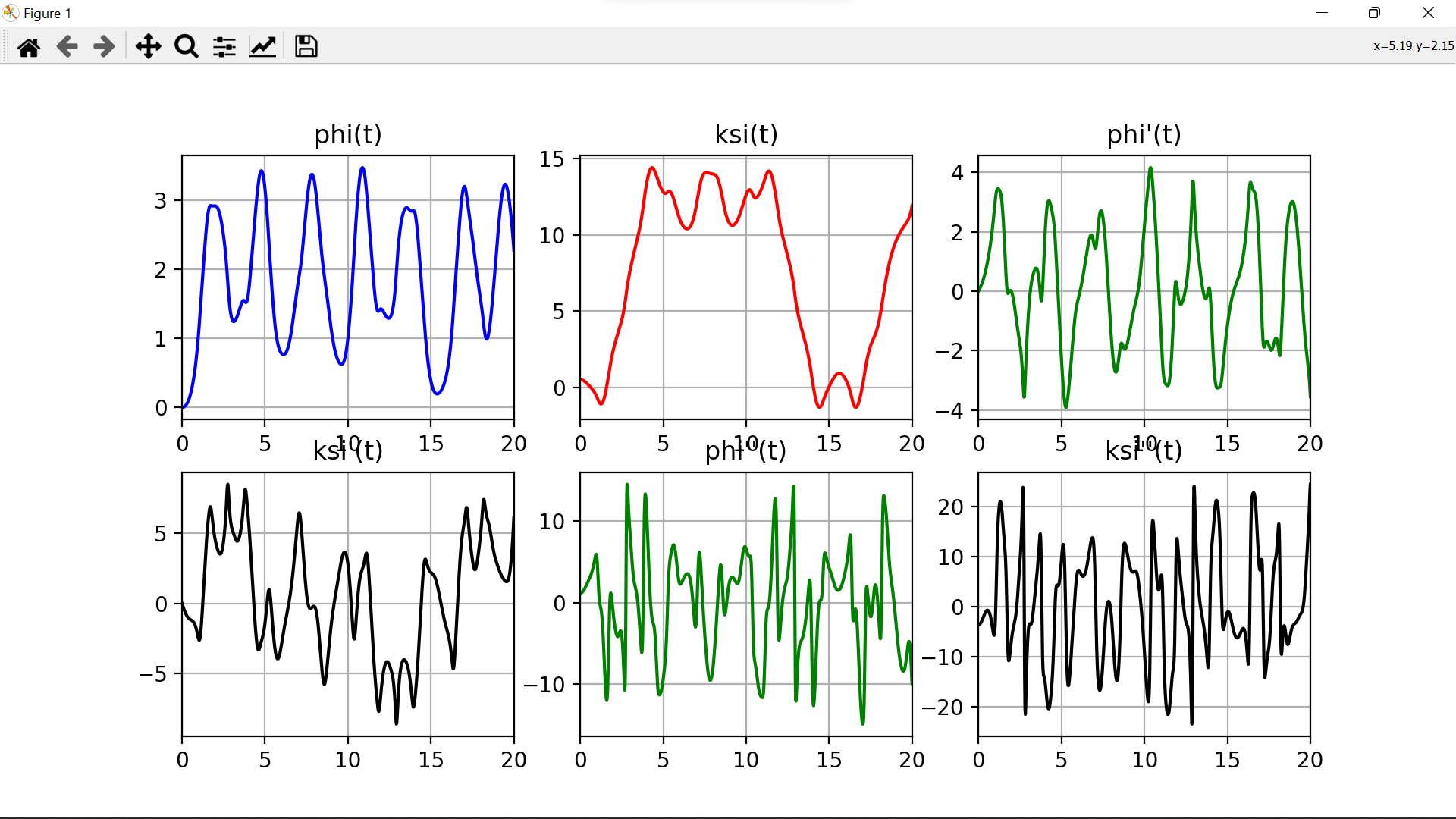
M1 = 5кг

М2 = 1кг

С = 10 Н\*м

r = 1 м

l = 1 м

**

При данных начальных условиях, система совершает затухающие колебания. Причем колебания с большим периодом.

1. Начальные условия

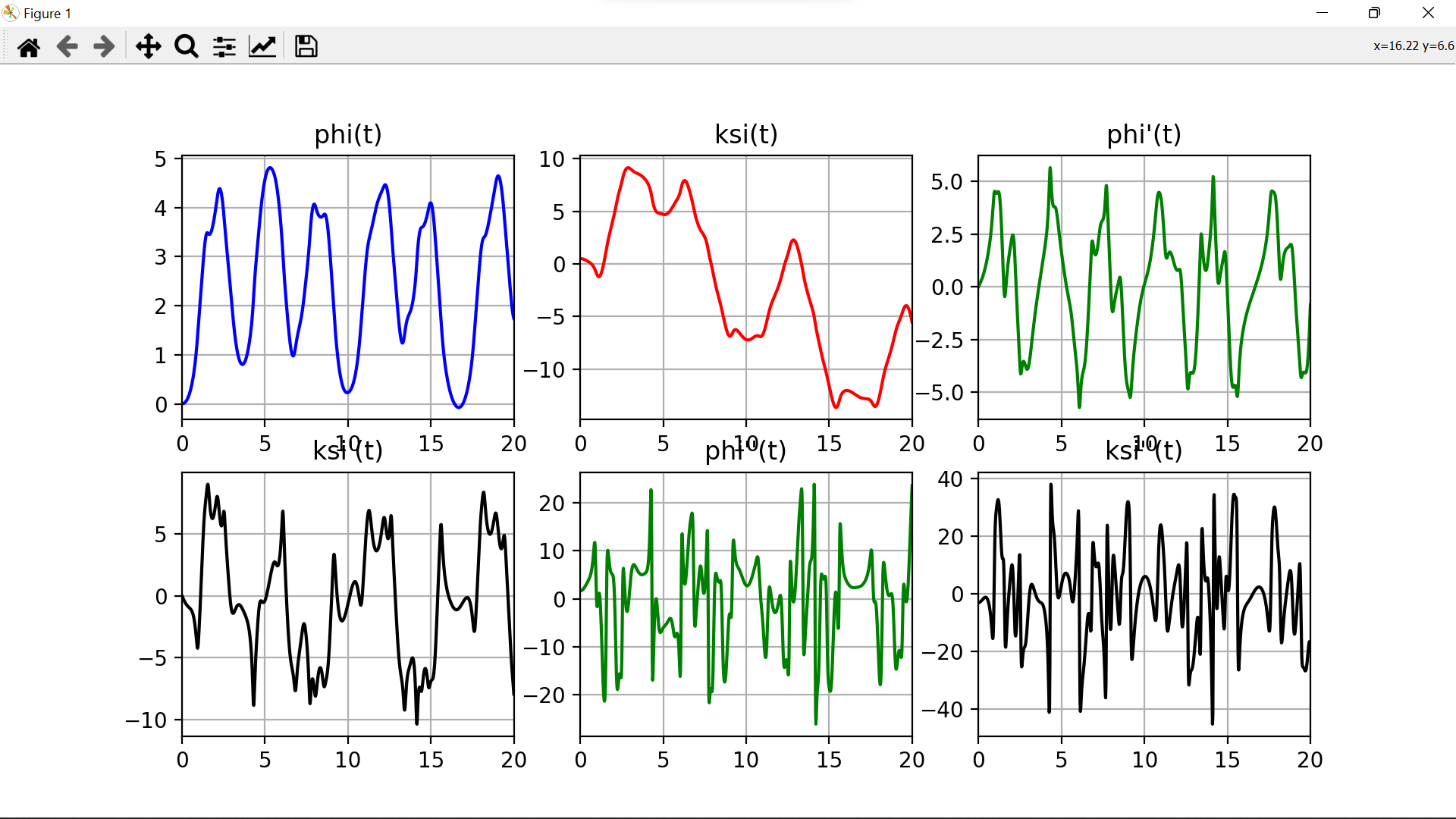
М1 = 10кг

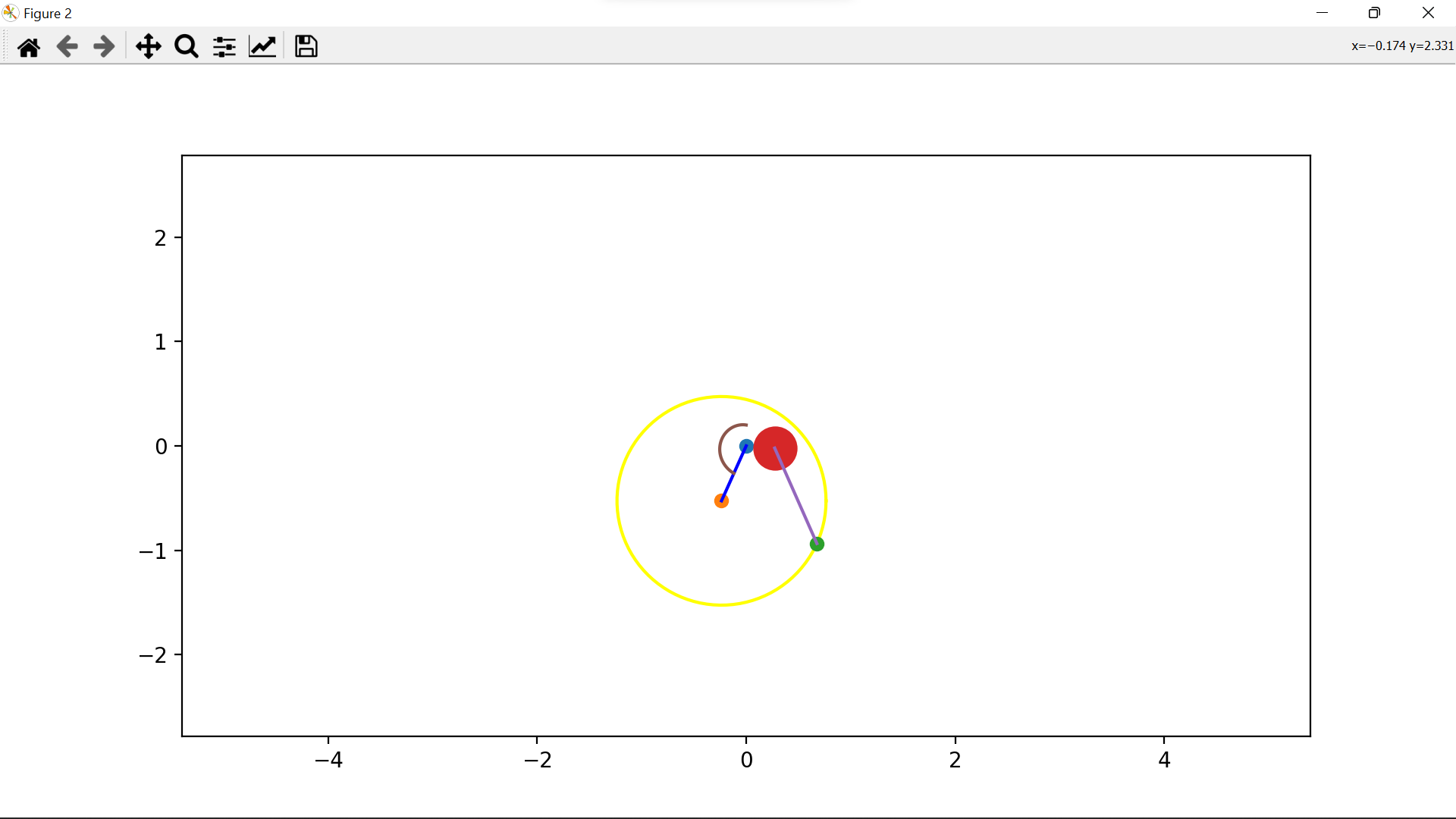
М2 = 3кг

С = 1 Н\*м

r = 1 м

l = 1 м





Можно заметить, что с увеличением массы и понижением жесткости пружины уменьшилась амплитуда колебаний грузика, при этом колебания амплитуда колебаний диска увеличилась

1. Начальные условия

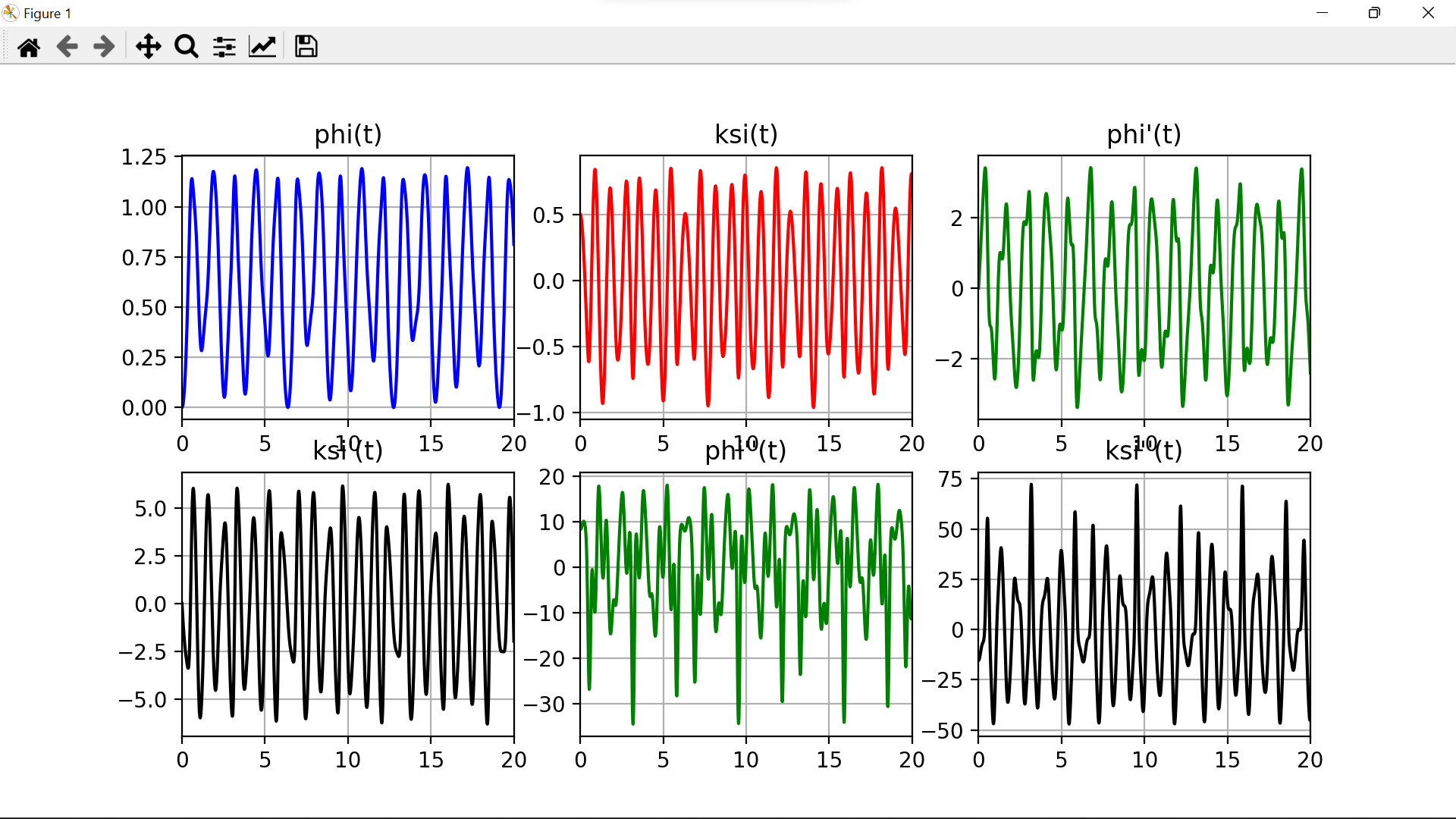
М1 = 10кг

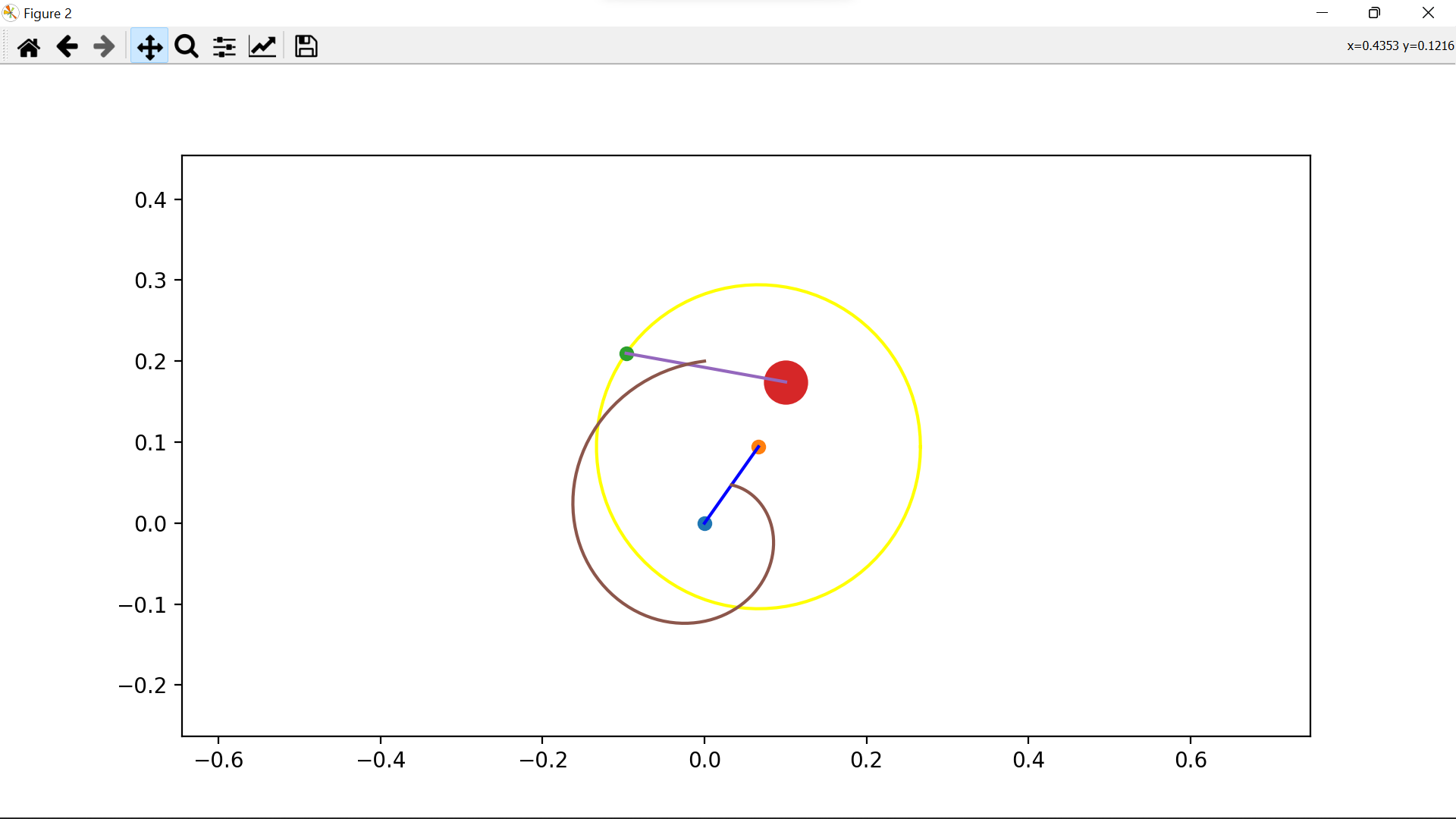
М2 = 3кг

С = 20 Н\*м

r = 0.2 м

l = 0.2 м





Изменение длины стержня и оси сильно сказалось на характере колебаний. Наша система стала совершать малые колебания.

1. Начальные условия

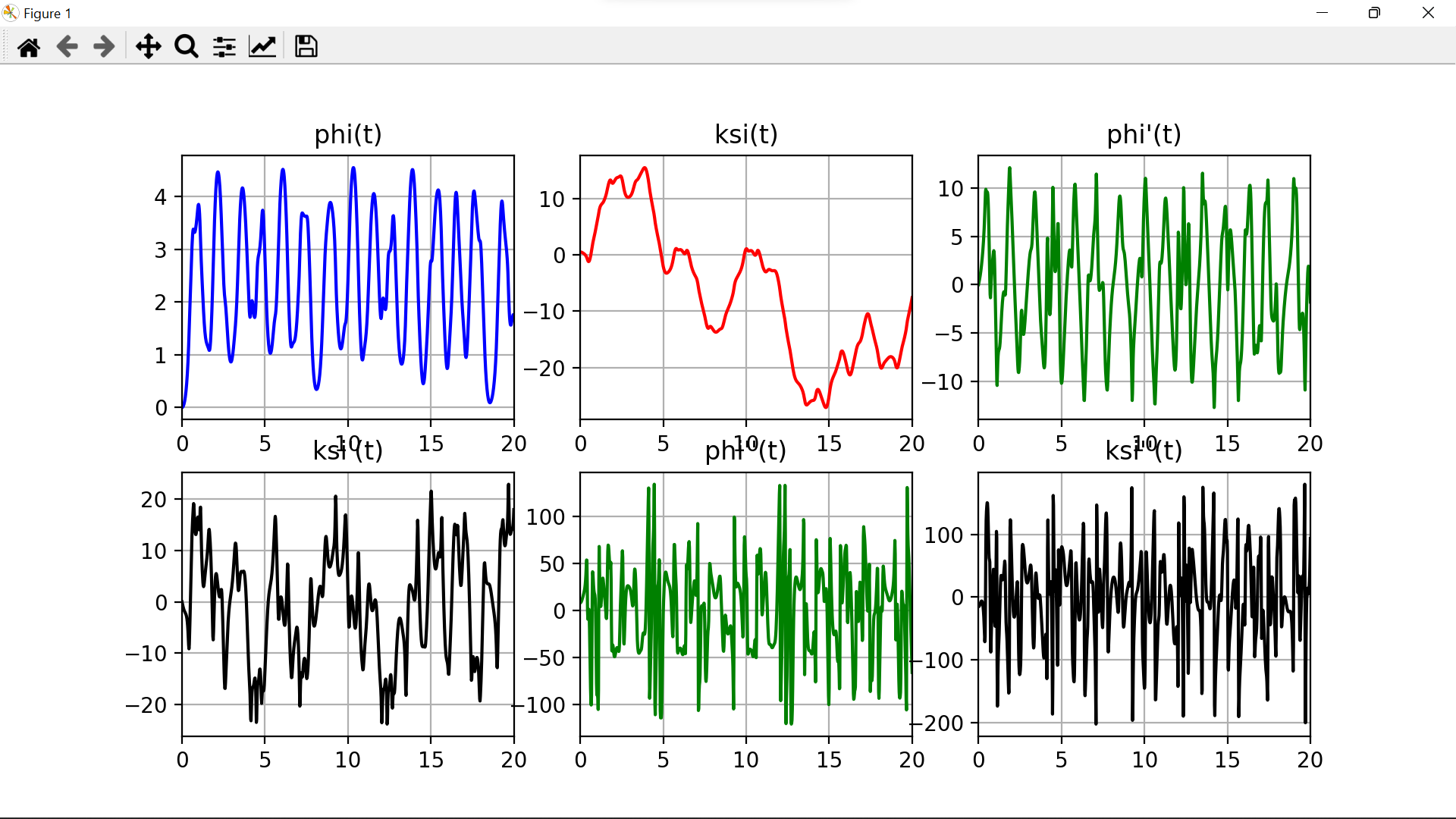
М1 = 10кг

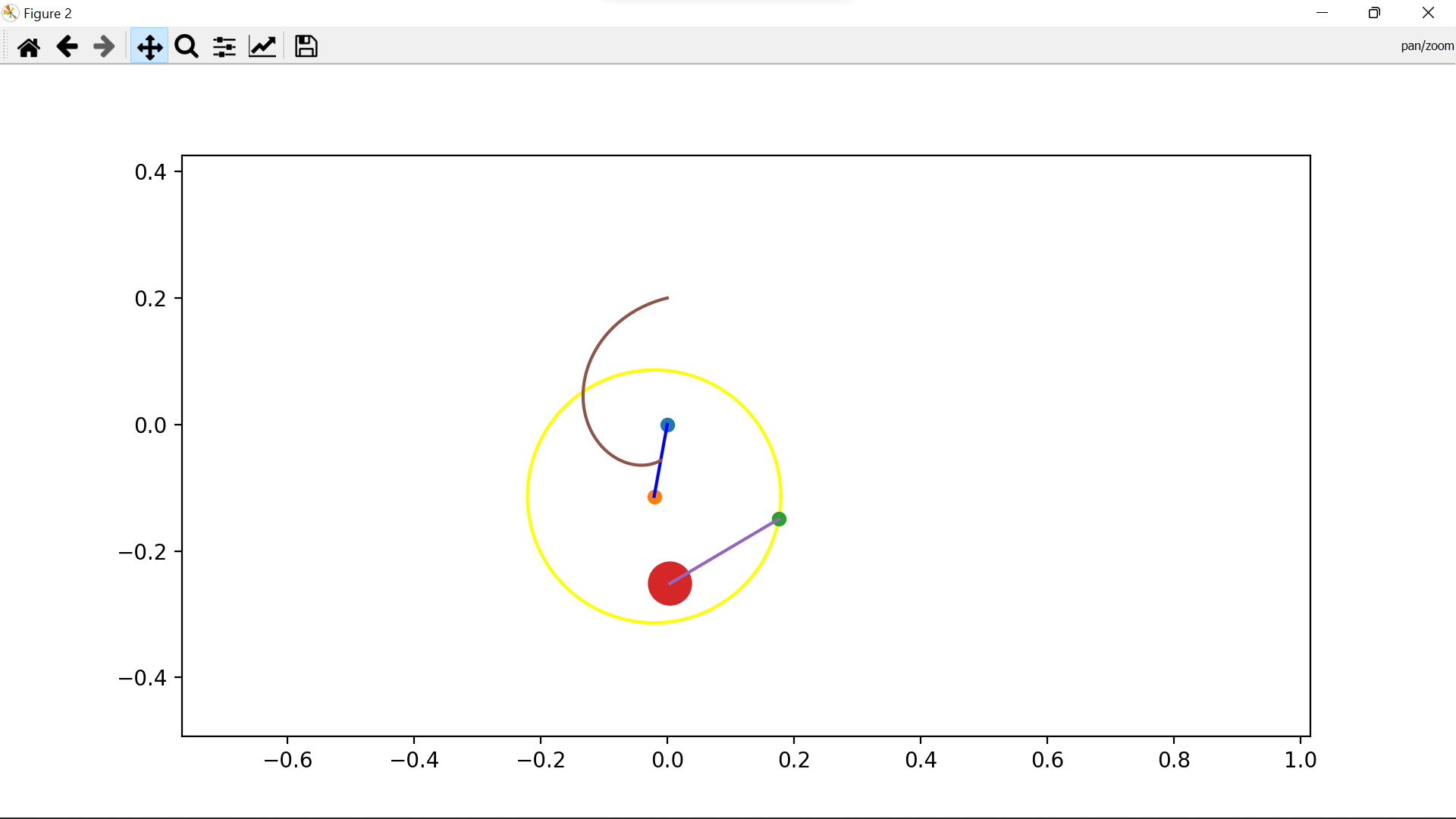
М2 = 3кг

С = 1 Н\*м

r = 0.2 м

l = 0.2 м





При уменьшении жесткости пружины, амплитуда системы резка возросла и она перестала совершать малые колебания.

Вывод: В результате данной лабораторной работы мы изучили данную нами систему, составили уравнения Лагранжа 2 рода, посмотрели на поведение системы при различных начальных условиях и изучили её устойчивость.

*Вычисления:*

