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 №3**

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

Белоусов Егор Леонидович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

1. **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 .

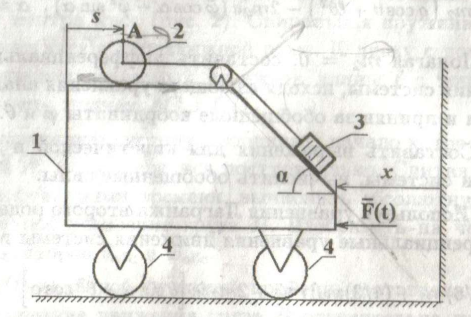
В основной программе требуется задать значения всех параметров, начальное положение системы, и запустить процедуру численного интегрирования системы.

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

1. **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:

Y = odeint(odesys, y0, t, (BoxM, CylM, WeightM, WheelM, Alpha, g))

1. **Численное интегрирование системы уравнений.**

Численное интегрирование системы дифференциальных уравнений в указанной выше форме производится с помощью функции odesys. Обращение к ней делается следующим образом:

odesys(y, t, BoxM, CylM, WeightM, WheelM, Alpha, g).  
BoxM – масса тележки;

CylM – масса цилиндра;

WeightM – масса груза;

WheelM – масса колеса;

Alpha – угол наклонной плоскости тележки с горизонтом;

g – ускорение свободного падения на Земле;

t – массив моментов времени, в который требуется дать ответ;

y – массив, в котором будут храниться значения ;

В начале основной программы запишем:

def odesys(y, t, M1, M2, M3, M4, Alpha, g):

dy = np.zeros(4)

dy[0] = y[2]

dy[1] = y[3]

a11 = M1 + M2 + M3 + 6 \* M4

a12 = -(M2 + M3 \* np.cos(Alpha))

a21 = -(M2 + M3 \* np.cos(Alpha))

a22 = M3 + 3 / 2 \* M2

b1 = 3 \* np.sin(0.5 \* t)

b2 = M3 \* g \* np.sin(Alpha)

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

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

return dy

1. **Построение графиков.**

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

N = [(BoxM + CylM + WeightM + 4 \* WheelM) \* g - WeightM -

np.sin(Alpha) \* odesys(y, t, BoxM, CylM, WeightM, WheelM, Alpha, g)[3]

for y, t in zip(Y, t)]

N1 = [g \* np.cos(Alpha) - np.sin(Alpha) \* odesys(y, t, BoxM, CylM, WeightM,

WheelM, Alpha, g)[2] for y, t in zip(Y, t)]

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

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

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

ax\_for\_graphs.set\_title('x(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, s, color='red')

ax\_for\_graphs.set\_title('s(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, N, color='green')

ax\_for\_graphs.set\_title('N(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, N1, color='black')

ax\_for\_graphs.set\_title('N1(t)')

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

ax\_for\_graphs.grid(True)

1. **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. Анимация:

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

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

ax.axis('equal')

ax.set(xlim=[-5, 20], ylim=[-4, 10])

ax.plot(X\_Ground, Y\_Ground, color='black', linewidth=3)

Drawed\_Wheel1 = ax.plot(X\_C1[0] + X\_Wheel, Y\_C1 + Y\_Wheel)[0]

Drawed\_Wheel2 = ax.plot(X\_C2[0] + X\_Wheel, Y\_C2 + Y\_Wheel)[0]

Drawed\_Block = ax.plot(X\_C3[0] + X\_Block, Y\_C3 + Y\_Block)[0]

Drawed\_Cyl = ax.plot(X\_A[0] + X\_Cyl, Y\_A + Y\_Cyl)[0]

Drawed\_Box = ax.plot(X\_O[0] + X\_Box, Y\_O + Y\_Box)[0]

Drawed\_Weight = ax.plot(X\_D[0] + X\_Weight, Y\_D[0] + Y\_Weight)[0]

Line\_A\_B1 = ax.plot([X\_A[0], X\_B1[0]], [Y\_A, Y\_B1], color=Drawed\_Block.get\_color())[0]

Line\_B2\_F = ax.plot([X\_B2[0], X\_F[0]], [Y\_B2, Y\_F[0]], color=Drawed\_Block.get\_color())[0]

Point\_A = ax.plot(X\_A[0], Y\_A, marker='o', markersize=5, color=Drawed\_Block.get\_color())[0]

AlphaWheel = -x / WheelR

Drawed\_WheelD1 = ax.plot([X\_C1[0] + WheelR \* np.sin(AlphaWheel[0]), X\_C1[0] - WheelR \* np.sin(AlphaWheel[0])],

[Y\_C1 + WheelR \* np.cos(AlphaWheel[0]), Y\_C1 - WheelR \* np.cos(AlphaWheel[0])])[0]

Drawed\_WheelD2 = ax.plot([X\_C2[0] + WheelR \* np.sin(AlphaWheel[0] + 1), X\_C2[0] - WheelR \* np.sin(AlphaWheel[0] + 1)],

[Y\_C2 + WheelR \* np.cos(AlphaWheel[0] + 1), Y\_C2 - WheelR \* np.cos(AlphaWheel[0] + 1)])[0]

def anima(i):

global s\_fixed

if (X\_A[i] + CylR > X\_B3[i]):

X\_A[i] = X\_B3[i] - CylR

if not s\_fixed:

s\_fixed = s[i]

X\_D[i] = X\_O[i] + BoxWUp + np.cos(Alpha) \* (InclStart + s\_fixed)

Y\_D[i] = Y\_O + BoxH - np.sin(Alpha) \* (InclStart + s\_fixed)

X\_F[i] = X\_D[i] + WeightH / 2 \* np.sin(Alpha)

Y\_F[i] = Y\_D[i] + WeightH / 2 \* np.cos(Alpha)

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

Line\_A\_B1.set\_data([X\_A[i], X\_B1[i]], [Y\_A, Y\_B1])

Line\_B2\_F.set\_data([X\_B2[i], X\_F[i]], [Y\_B2, Y\_F[i]])

Drawed\_Box.set\_data(X\_O[i] + X\_Box, Y\_O + Y\_Box)

Drawed\_Cyl.set\_data(X\_A[i] + X\_Cyl, Y\_A + Y\_Cyl)

Drawed\_Block.set\_data(X\_C3[i] + X\_Block, Y\_C3 + Y\_Block)

Drawed\_Weight.set\_data(X\_D[i] + X\_Weight, Y\_D[i] + Y\_Weight)

Drawed\_Wheel1.set\_data(X\_C1[i] + X\_Wheel, Y\_C1 + Y\_Wheel)

Drawed\_Wheel2.set\_data(X\_C2[i] + X\_Wheel, Y\_C2 + Y\_Wheel)

Drawed\_WheelD1.set\_data([X\_C1[i] + WheelR \* np.sin(AlphaWheel[i]), X\_C1[i] - WheelR \* np.sin(AlphaWheel[i])], [Y\_C1 + WheelR \* np.cos(AlphaWheel[i]), Y\_C1 - WheelR \* np.cos(AlphaWheel[i])])

Drawed\_WheelD2.set\_data([X\_C2[i] + WheelR \* np.sin(AlphaWheel[i] + 1), X\_C2[i] - WheelR \* np.sin(AlphaWheel[i] + 1)], [Y\_C2 + WheelR \* np.cos(AlphaWheel[i] + 1), Y\_C2 - WheelR \* np.cos(AlphaWheel[i] + 1)])

return [Point\_A, Drawed\_Box, Drawed\_Wheel1, Drawed\_Wheel2, Drawed\_WheelD1, Drawed\_WheelD2]

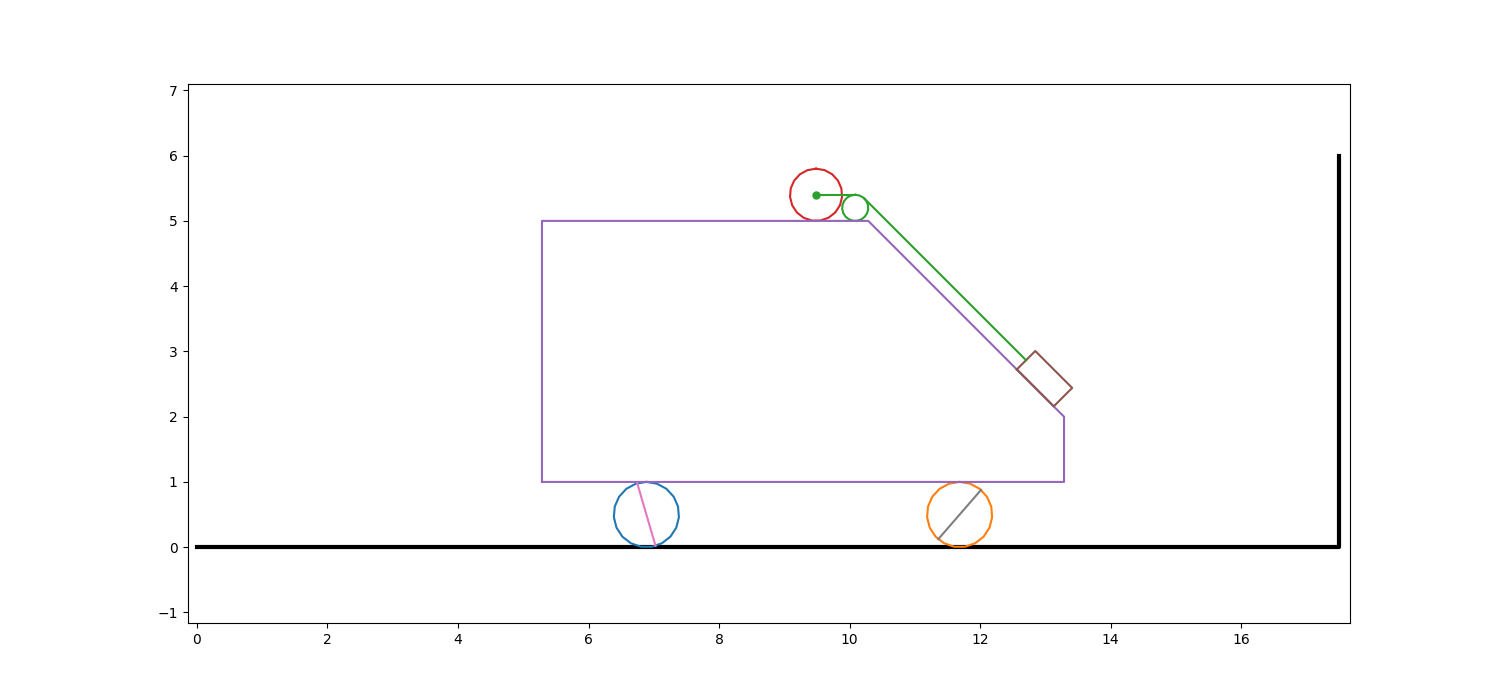
anim = FuncAnimation(fig, anima, frames=len(t), interval=10)

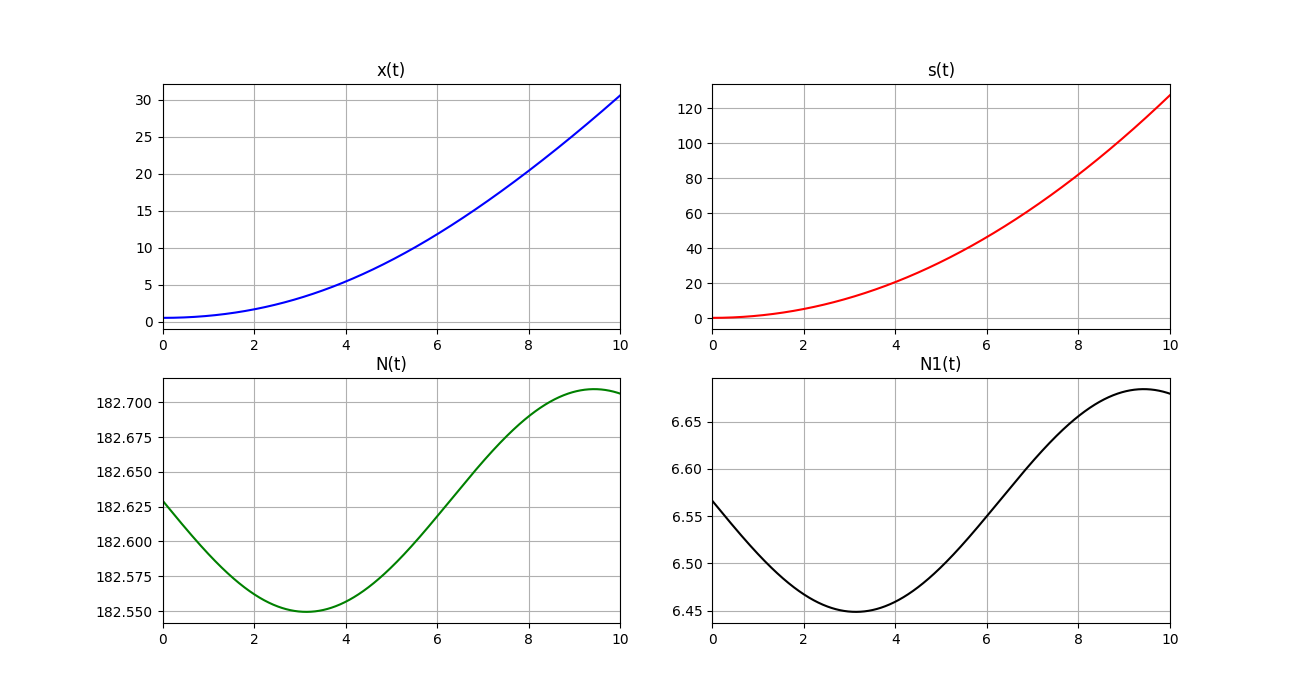
plt.show()

1. **Результат работы программы.**

Выведем полученные графики работы программы:

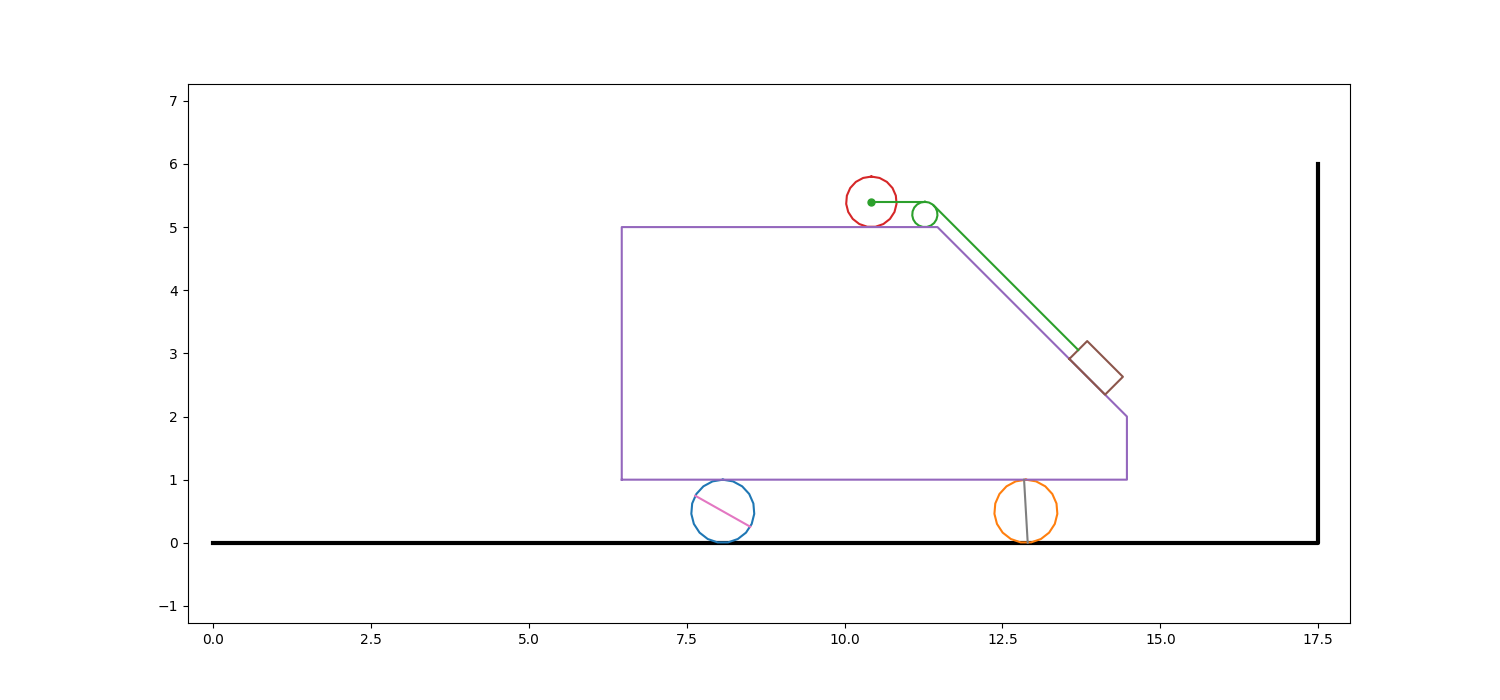
1. BoxM = 10; CylM = 3; WeightM = 2; Wheel = 1; Alpha = pi / 4; g = 9.81. Стандартная ситуация (нормальные условия):

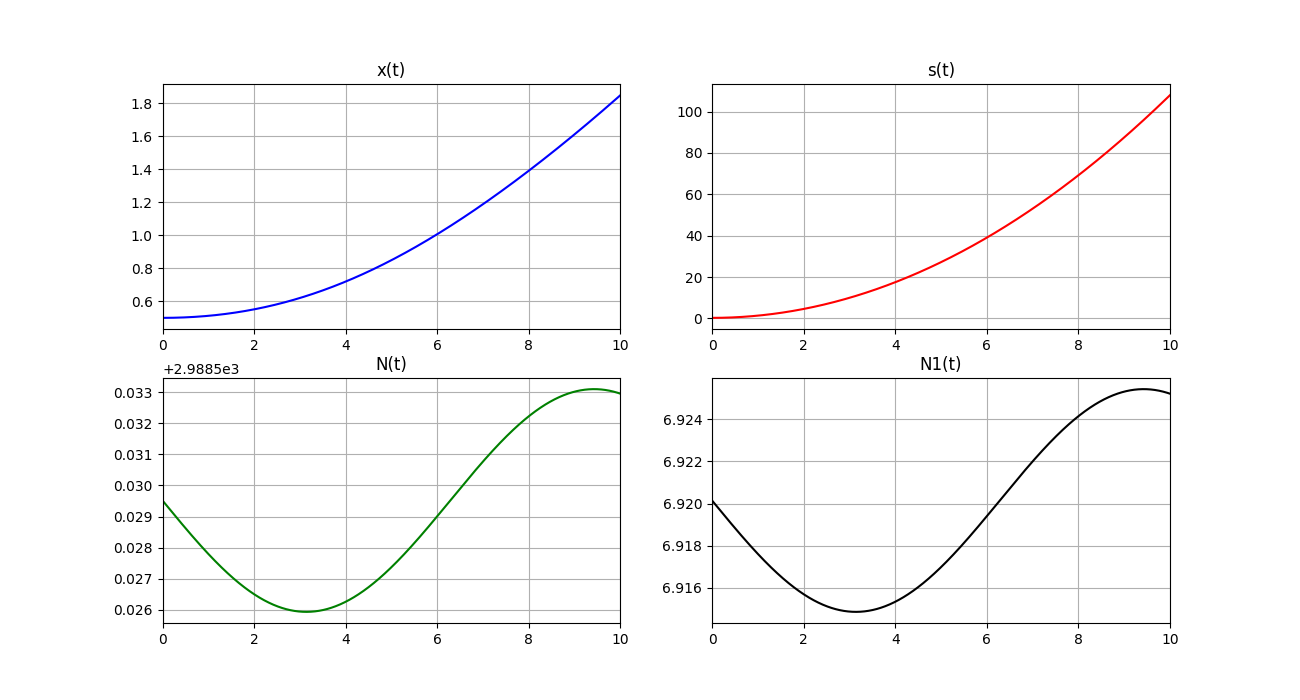




Результат: тележка медленно едет влево, грузик скатывается по наклонной плоскости, цилиндр, связанный нитью с грузиком, движется вслед за ним. Достигнув блока, перекрывающего путь цилиндру, он останавливается, как и грузик на наклонной плоскости.

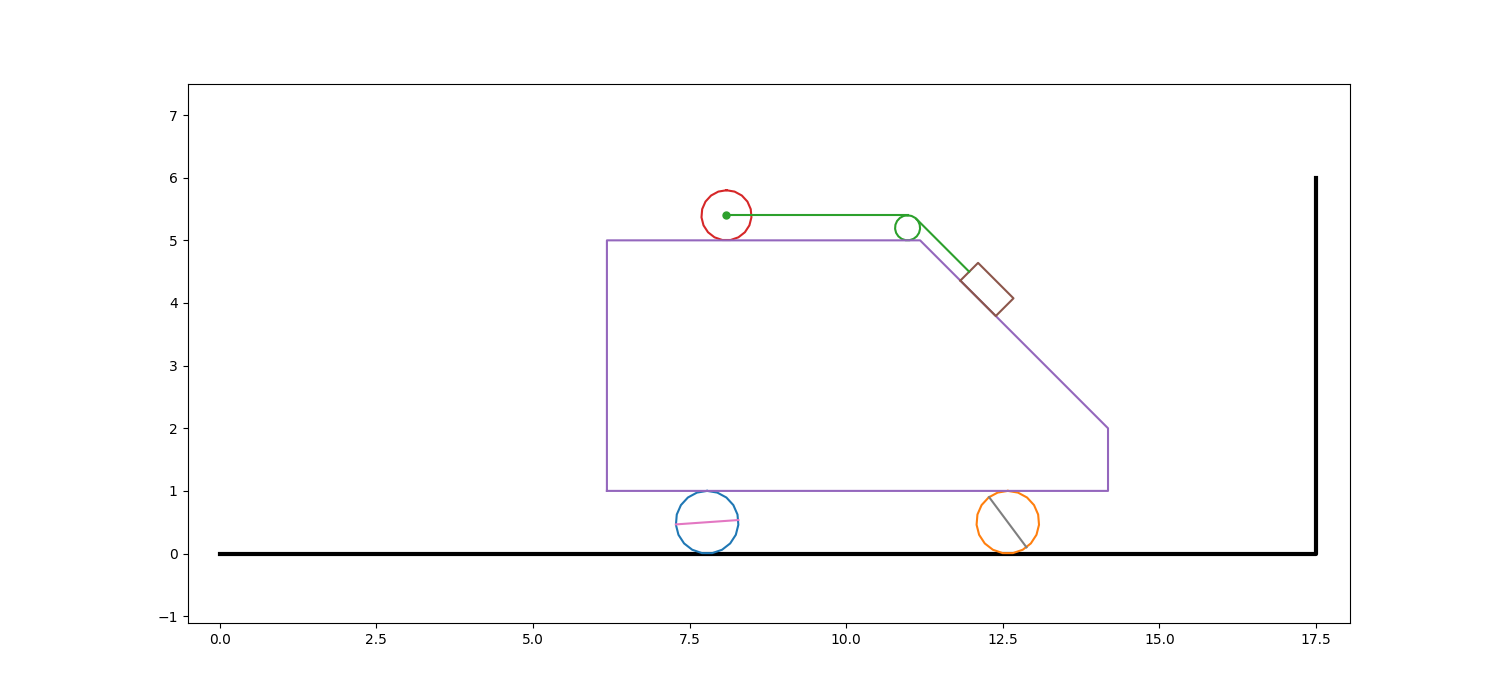
1. BoxM = 100; CylM = 3; WeightM = 2; Wheel = 50; Alpha = pi / 4; g = 9.81. Тележка и колеса большой массы:

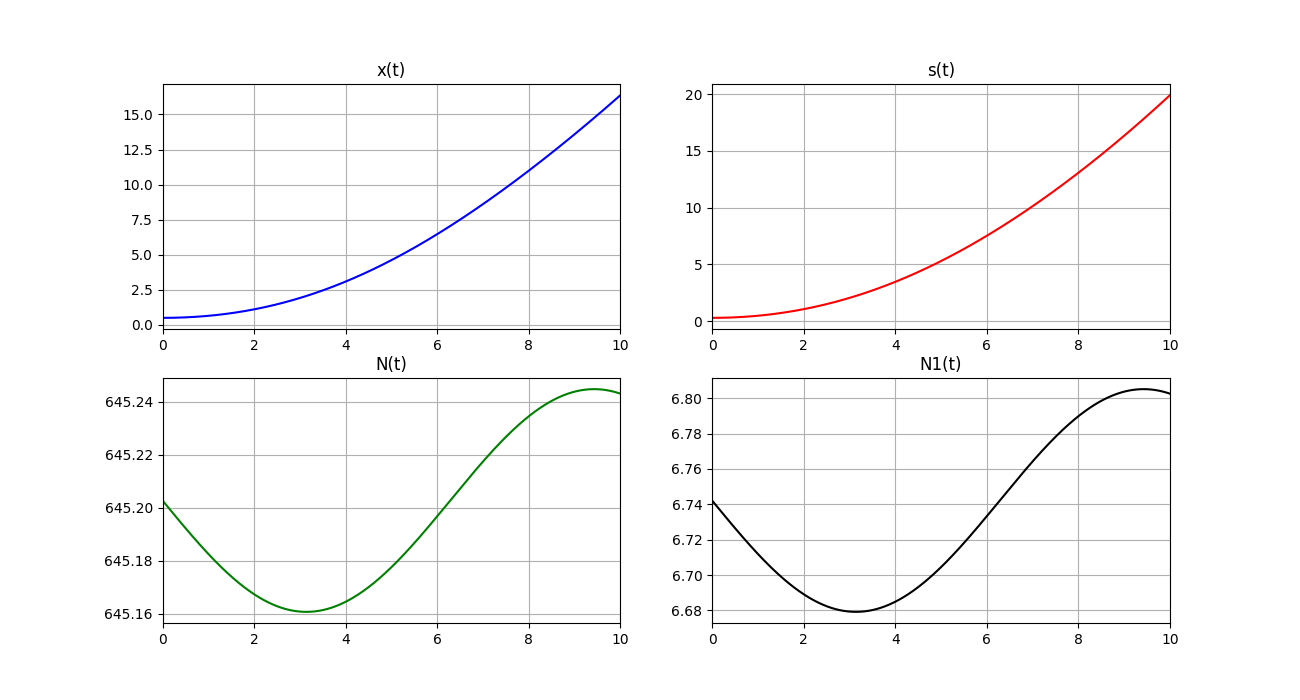




Результат: тележка очень медленно движется влево, цилиндр и грузик двигаются почти как обычно. По шкале значений оси игрек на графике x(t) видно, что он возрастает сильно медленнее, чем при нормальных значениях массы.

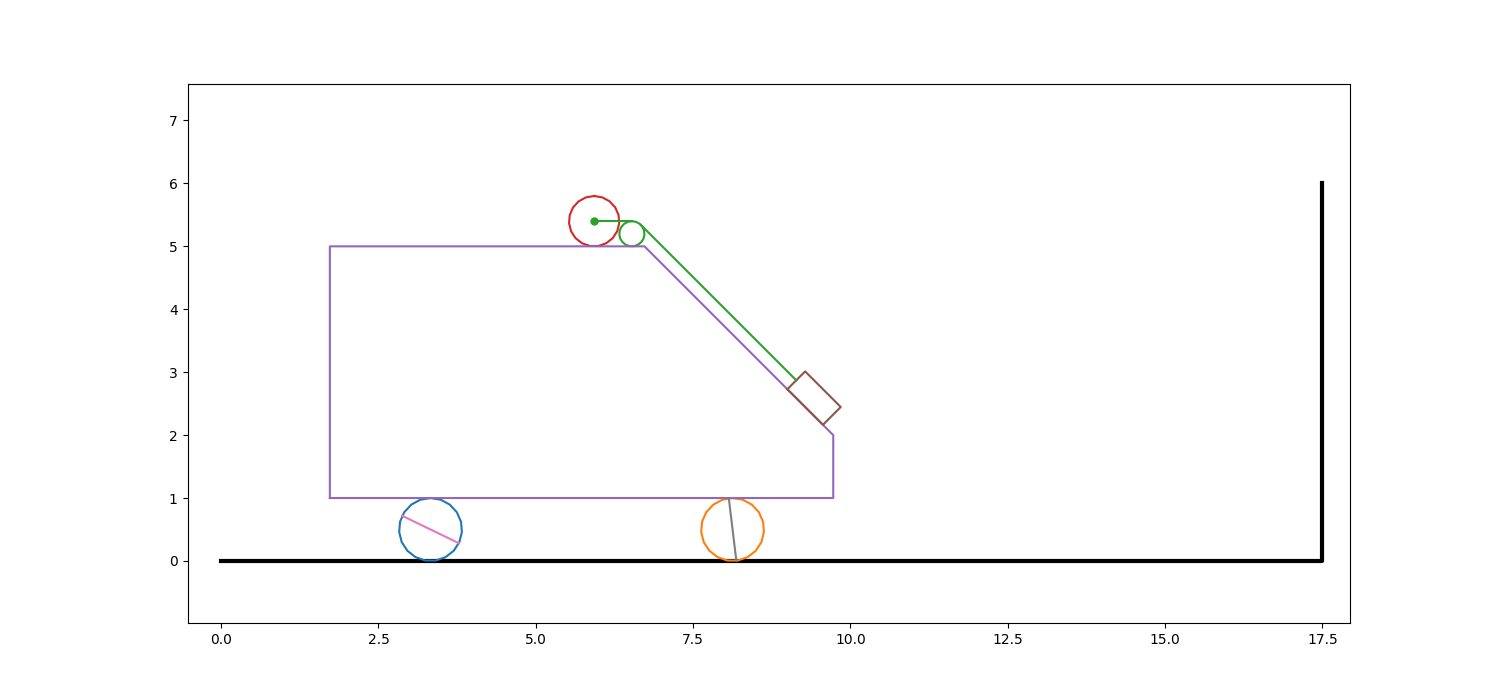
1. BoxM = 10; CylM = 50; WeightM = 2; Wheel = 1; Alpha = pi / 4; g = 9.81. Тяжелый цилиндр:

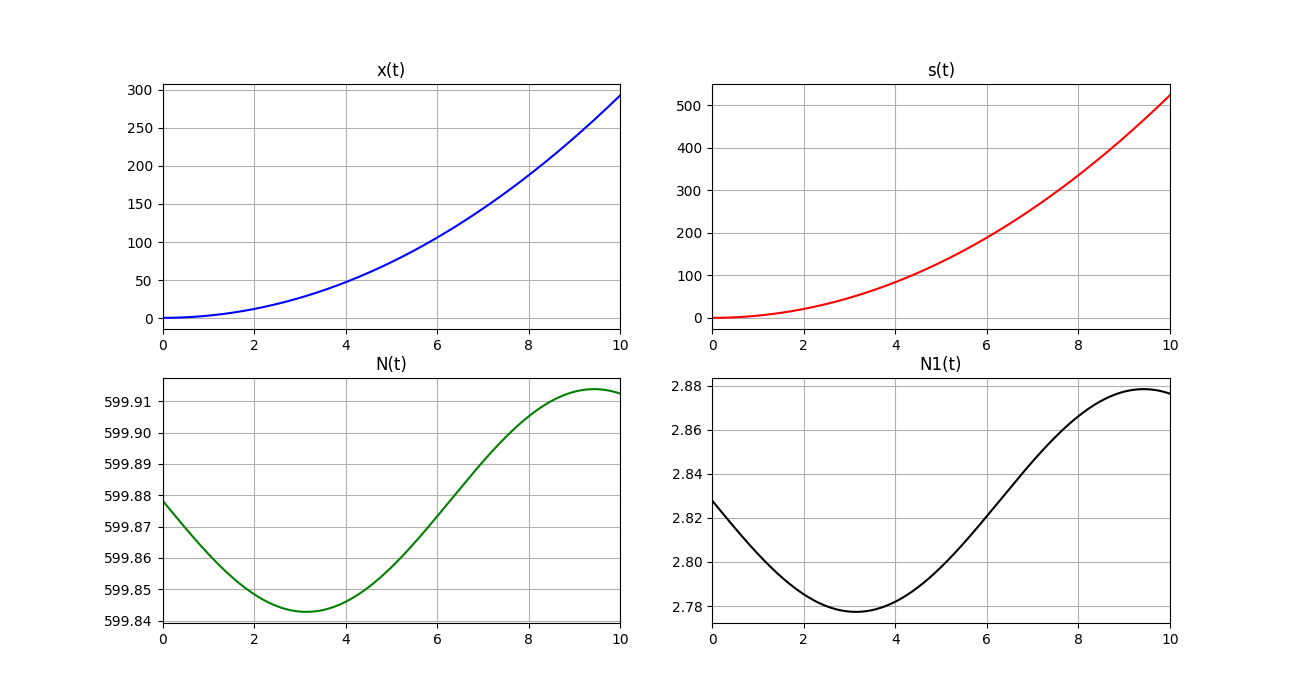




Результат: тележка едет влево с нормальной скоростью, грузик скатывается вниз вначале медленно, затем нормально, цилиндр передвигается крайне медленно. На графиках x(t) и s(t) видно сильное замедление их возрастания по сравнению с нормальными условиями.

1. BoxM = 10; CylM = 3; WeightM = 50; Wheel = 1; Alpha = pi / 4; g = 9.81. Тяжелый грузик:





Результат: грузик быстро съезжает по наклонной плоскости, цилиндр едет за ним до упора. Тележка быстро едет в левую сторону. На графиках x(t) и s(t) видно, что координата y увеличивается в разы быстрее, чем при нормальных условиях.