ИТМО

Моделирование физического процесса "Мертвая петля"

Группа M3203
Кравченкова Елизавета

Преподаватель
Хуснутдинова Наира Рустемовна

Физические основы компьютерных и сетевых технологий
Университет ИТМО
Санк-Петербург, Россия

7 января 2024 г.

Оглавление

Задание

Тело массой m разгоняется в горизонтальной плоскости и попадает на вертикально расположенный фрагмент кольца(дугу) радиуса R и угловым размером $\alpha(\frac{\pi}{2} <= a <= \frac{3\pi}{2})$. Определить начальную скорость тела, необходимую для прохождения всей длины дуги. Построить(визуализировать) траекторию тела после отрыва от дуги. Дуга имеет коэффициент трения μ. Программа должна предусматривать изменение параметров, приведенных ниже.

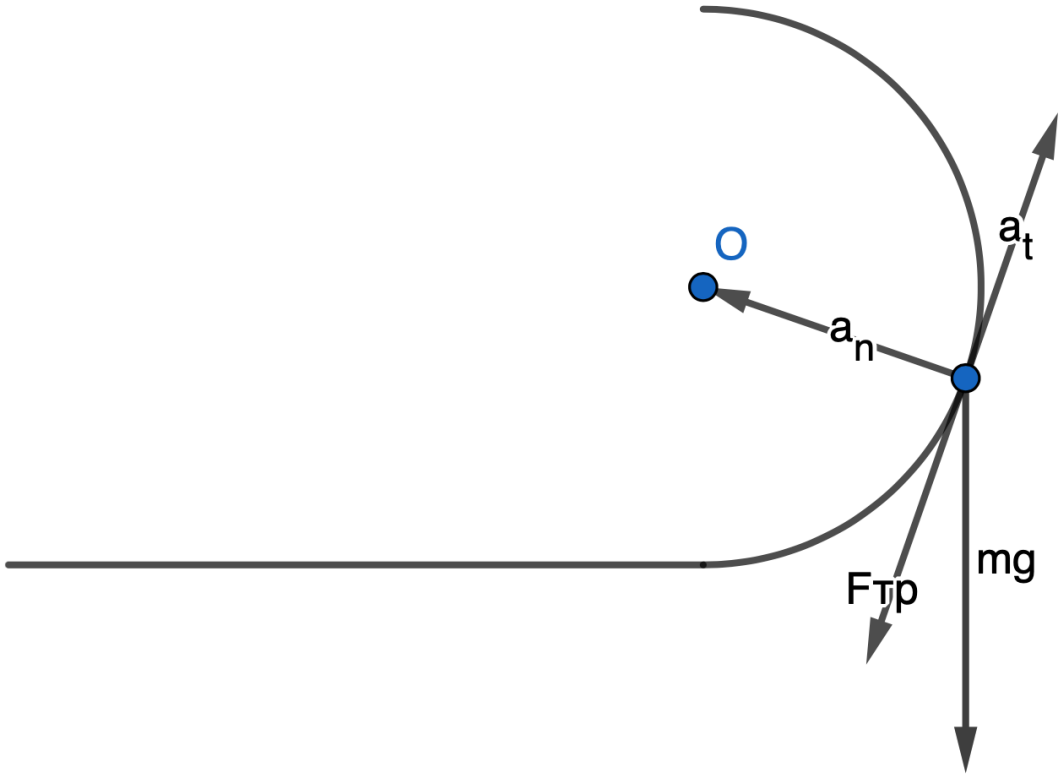

Рис. 1.1: Иллюстрация к задаче

Данные взять из таблицы. Номер варианта соответствует номеру по списку группы.
Вариант 9:
Масса тела $m = 3$, кг
Радиус кольца R = 5 м
Угловой размер дуги $\alpha = \frac{\pi}{2} + \frac{\pi}{6}$ рад
Коэффициент трения $\mu = 0.04$

1.1 Объект исследования

Исследование движения тела вверх по дуге окружности с учетом силы трения.

1.2 Метод экспериментального исследования

Имитация физической модели с помощью языка программирования

Цели и Задачи

Цель: Выполнить численное моделирование движения тела по дуге окружности и после отрыва от нее Найти минимальную начальную скорость, с которой объект пройдет всю длину дуги.

Задачи:

1. Вывести необходимые для моделирования формулы.
2. Имитация физической модели с помощью языка программирования python.
3. Исследование полученных результатов.

Теория и рабочие формулы

3.1 Первая часть пути: движение по прямой с заданным ускорением

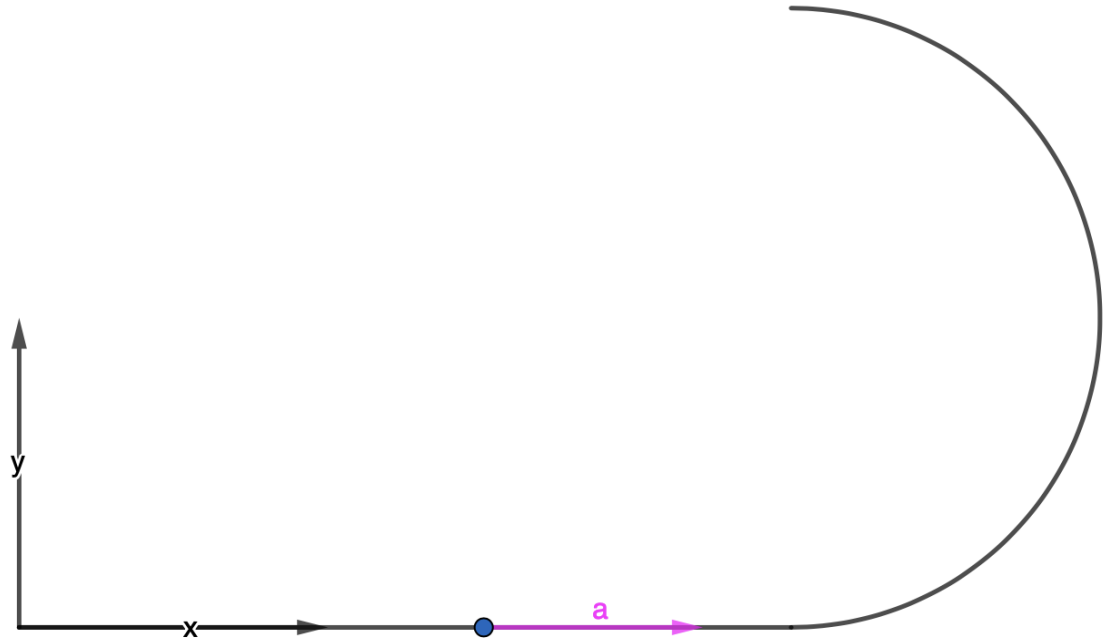

Рис. 3.1: Первая часть пути: движение по прямой с заданным ускорением a

Мы имеем дело с обычным равноускоренным движением:

$$v = v_0 + at$$

$$x = x_0 + v_0 t + \frac{at^2}{2}$$

3.2 Вторая часть пути: движение вверх по дуге

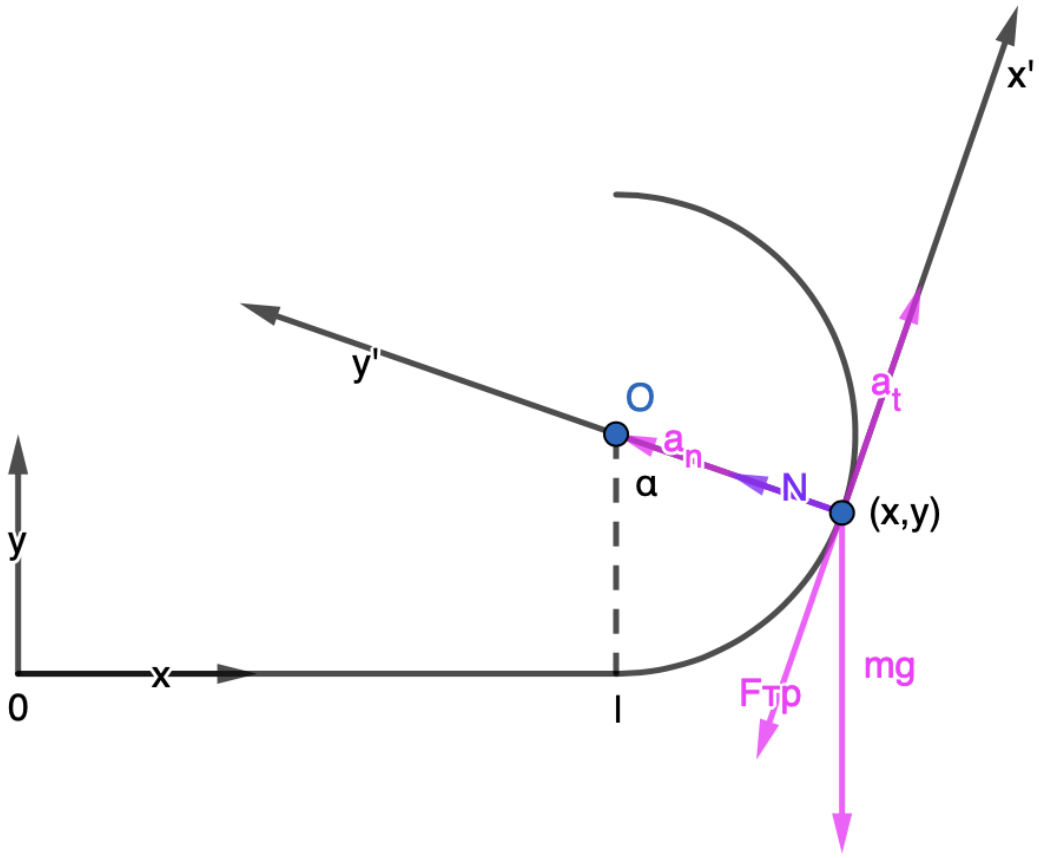

Рис. 3.2: Вторая часть пути: движение вверх по дуге

$$a_n = \frac{v^2}{R}$$

$$F_{\text{тр}} = \mu N$$

Рассмотрим проекции на Ox' и Oy':

$$Oy' : N - mg\cos\alpha = \frac{v^2}{R}m$$

$$Ox' : -\mu N - mg\sin\alpha = ma_t$$

Отсюда получаем формулы для a_t и N:

$$N = m(\frac{v^2}{R} + g\cos\alpha)$$

$$a_t = -\mu(\frac{v^2}{R} + g\cos\alpha) - g\sin\alpha$$

Если тело имеет координаты (x,y) мы можем определить текущий угол:

$$\cos\alpha = \frac{R-y}{R}$$

$$\alpha = \arccos\frac{R-y}{R}$$

Так как arccos принимает значения от 0 до π, то придется рассмотреть отдельный случай когда дуга больше, чем π:

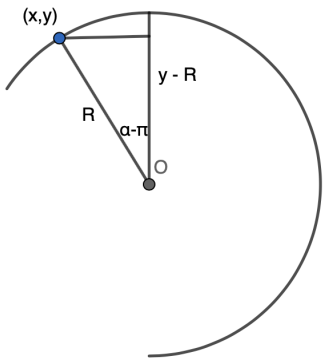

Рис. 3.3: положение тела больше пи

$$\cos(\alpha - \pi) = \frac{y-R}{R}$$

$$\alpha = \arccos\frac{y-R}{R} + \pi$$

Удобнее свести наши ускорения от "частной"отдельной для каждой точки системы Ox'y' к общей Oxy:

$$a_x = -\sin\alpha \cdot a_n + \cos\alpha \cdot a_t$$

$$a_y = \cos\alpha \cdot a_n + \sin\alpha \cdot a_t$$

Тело движется с переменныым ускорением по x и по y. В нашем моделировании Будут рассматриваться очень малые промежутки времени, в рамках которых ускорение постоянно.

$$\Delta v_x = a_x \Delta t$$

$$\Delta x = v_x \Delta t + \frac{a_x \Delta t^2}{2}$$

$$\Delta v_y = a_y \Delta t$$

$$\Delta y = v_y \Delta t + \frac{a_y \Delta t^2}{2}$$

Тело оторвется от дуги, когда сила реакции опоры N станет равной нулю или когда пройдет ее всю.

3.3 Третья часть пути: движение после отрыва от дуги

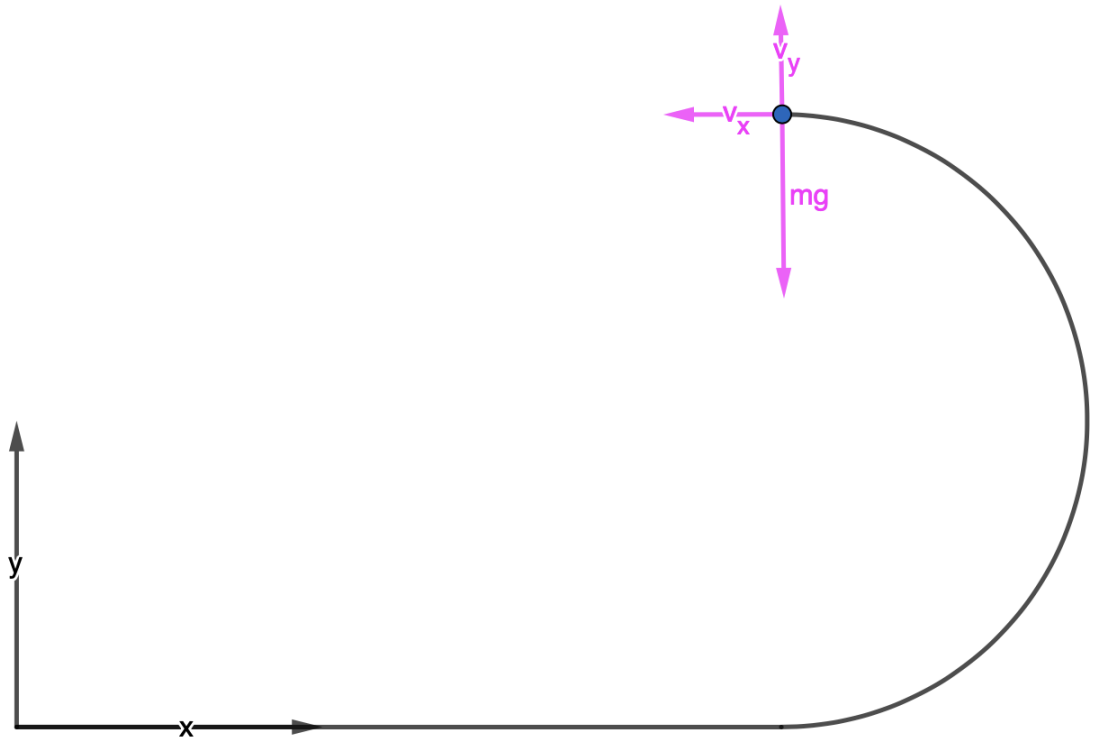

Рис. 3.4: Третья часть пути: движение после отрыва от дуги

После отрыва от дуги тело тело движется по баллистической траектории:

$$v_x = const$$

$$\Delta x = v_x \Delta t$$

$$v_y = v_y 0 - gt$$

$$\Delta y = v_y \Delta t - g \Delta t$$

3.4 Четвертая часть пути: движение по дуге окружности вниз

Если получилось так, что при падении тело упало на дугу, то законы перемещения следующие:

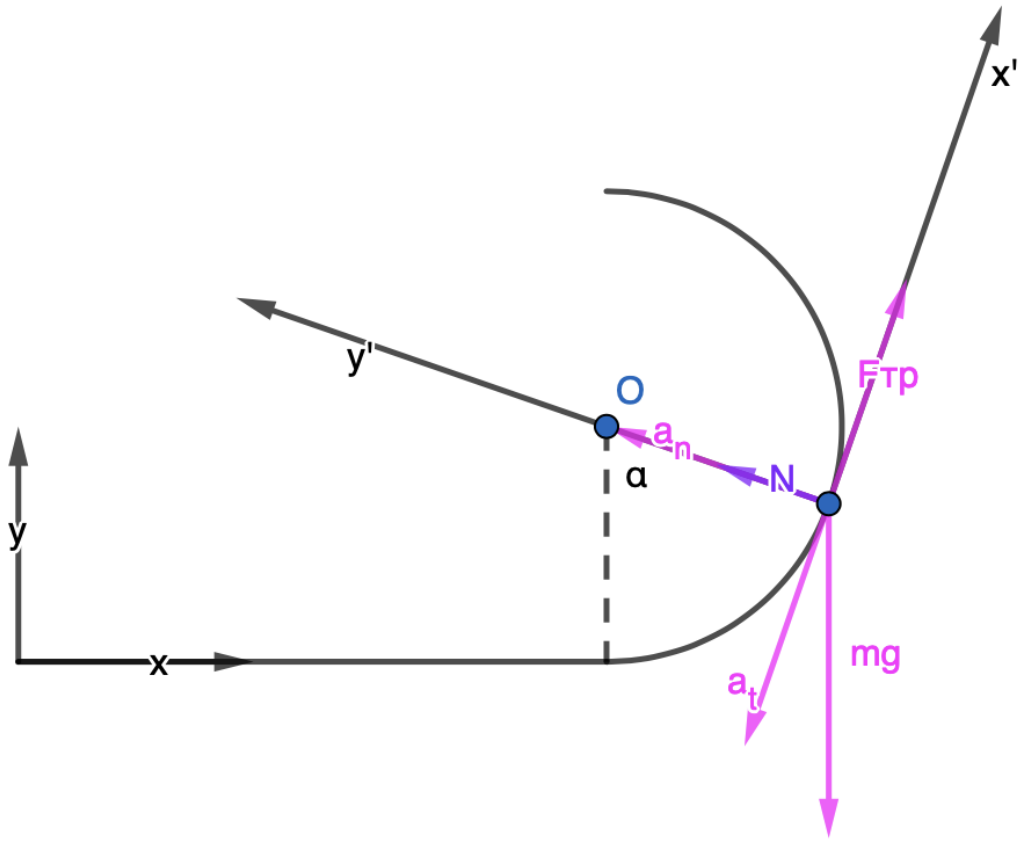

Рис. 3.5: Четвертая часть пути: движение по дуге окружности вниз

$$a_n = \frac{v^2}{R}$$

$$F_{\text{тр}} = \mu N$$

Рассмотрим проекции на Ox' и Oy':

$$Oy' : N - mg\cos\alpha = \frac{v^2}{R}m$$

$$Ox' : \mu N - mg\sin\alpha = -ma_t$$

Отсюда получаем формулы для a_t и N:

$$N = m(\frac{v^2}{R} + g\cos\alpha)$$

$$a_t = g\sin\alpha - \mu(\frac{v^2}{R} + g\cos\alpha)$$

Если тело имеет координаты (x,y) мы можем определить текущий угол определяем также как и в пункте с подъемом:

Найдем проекции на Oxy:

$$a_x = -\sin\alpha \cdot a_n - \cos\alpha \cdot a_t$$

$$a_y = \cos\alpha \cdot a_n - \sin\alpha \cdot a_t$$

Тело движется с переменныым ускорением по x и по y. В нашем моделировании Будут рассматриваться очень малые промежутки времени, в рамках которых ускорение постоянно.

$$\Delta v_x = a_x \Delta t$$

$$\Delta x = v_x \Delta t + \frac{a_x \Delta t^2}{2}$$

$$\Delta v_y = a_y \Delta t$$

$$\Delta y = v_y \Delta t + \frac{a_y \Delta t^2}{2}$$

Ход работы

4.1 Поготовка

Для визуализации физической модели был выбран язык python. Для работы с графическим интерфейсом использовалась библиотека pygame. Для ее установки пропишем в терминал

```
pip install pygame
```

4.2 Основная часть

4.2.1 Задание констант

Зададим константы для нашего приложения: размеры окна, цвета, шрифты и физические данные.

```
#constants
WIDTH, HEIGHT = 800, 700
FPS : Final = 60

modelColor : Final = pg.Color('black')
textColor : Final = pg.Color('black')
objectColor : Final = (73,16,139)
buttonNoactive : Final = (226,110,229)
buttonActive : Final = (126,48,225)
windowCol : Final = pg.Color('#F3F8FF')
color_inactive : Final = pg.Color('#E26EE5')
color_active : Final = pg.Color('#49108B')
font : Final = pg.font.Font(None, 32)
font2 : Final = pg.font.Font(None, 20)

#-----

# data for visualizing the model
distPix: Final = 300
posStart: Final = 150
heigh : Final = 400
mnozh : Final = 20
rad : Final = 5
```

```
24
25 #-----
26
27 # physical parameters
28 g : Final = 9.8
29 m : Final = 3
30 R : Final = 5
31 toch : Final = 1000
32 alpha : Final  = 4*math.pi/6
33 mu : Final  = 0.04
```

4.2.2 Создание модели

Создадим класс Model, который будет хранить текущее положение тела, массивы позиций по x и y, а также другие массивы: v,a,t. Также этот класс при создании экземляра с помощью алгоритма бинарного поиска подбирает такое минимальное значение v_0, при котором тело проходит всю дугу.(но об этом позже). Еще этот класс имеет метод DO, именно он выполняет всю логику по перемещению тела(об этом тоже позже). Вообщем наш класс обладает следующей структурой:

```
1  class Model:
2      v_0 = 0
3      v_0_ans = 0
4      a_0 = 0
5      start_pos_x = 0
6      start_pos_y = 0
7      dist = 0
8
9      cur_v = 0
10     cur_t = 0
11     pos_x = 0
12     pos_y = 0
13     cur_a = 0
14     cur_ax = 0
15     cur_ay = 0
16     cur_vx = 0
17     cur_vy = 0
18
19     t_ = []
20     x_ = []
```

```
    y_ = []
    v_ = []
    a_ = []

    def __init__(self,start_pos_x, start_pos_y, dist,v_0, a = 0):
        # ---
        v_0_ans = self.BinPoisk()

    def clean(self)
    def FirstPath(self)
    def SecondPath(self)
    def ThirdPath(self)
    def fourthPath(self)
    def DO(self)
```

4.2.3 Реализация основных функций физической модели

Как только мы создали экземпляр нашей модельки, ей надо присвоить какое-нибудь v_0. После этого можно запускать метод Do. Он посчитает все нужные нам значения. Если частица пролетела всю дугу, метод вернет true, иначе false.
Do():

```
    def DO(self):
        self.clean()
        self.FirstPath()
        allWay = self.SecondPath()
        self.ThirdPath()
        return allWay
```

Далее рассмотрим метод **FirstPath()**, моделирующий первую часть пути:

```
        def FirstPath(self):
            dt =1/self.toch
            self.cur_a = self.a_0
            while (self.pos_x <(self.dist+self.start_pos_x)):
                self.t_.append(self.cur_t)
                self.a_.append(self.cur_a)
                self.v_.append(self.cur_v)
                self.x_.append(self.pos_x)
                self.y_.append(self.pos_y)
```

```
10
11
12              self.pos_x = self.start_pos_x + self.cur_v*self.cur_t +  self.
    cur_a*self.cur_t*self.cur_t/2
13              self.pos_y = self.start_pos_y
14              self.cur_v = self.v_0 + self.cur_a*self.cur_t
15              self.cur_t += dt
16          self.cur_vx = self.cur_v
17          self.cur_vy = 0
```

Далее рассмотрим метод **SecondPath()**, моделирующий вторую часть пути:

```
1       def SecondPath(self):
2           dt =1/self.toch
3           allWay= False
4           alpha_cur = 0
5           while (not fail):
6
7               a_c= self.cur_v**2/self.R
8               a_t = -self.mu * (a_c + self.g*math.cos(alpha_cur)) - self.g*math.
    sin(alpha_cur)
9
10              self.cur_ay = math.cos(alpha_cur)*a_c +math.sin(alpha_cur)*(a_t)
11              self.cur_ax = -math.sin(alpha_cur)*a_c +math.cos(alpha_cur)*(a_t)
12              self.cur_a = math.sqrt(self.cur_ax**2 + self.cur_ay**2)
13
14              self.cur_vx += self.cur_ax*dt
15              self.cur_vy += self.cur_ay*dt
16              self.cur_v = math.sqrt(self.cur_vx**2 + self.cur_vy**2)
17
18              self.pos_x = self.pos_x +self.cur_vx*dt + self.cur_ax*dt*dt/2
19              self.pos_y = self.pos_y + self.cur_vy*dt + self.cur_ay*dt*dt/2,
    self.start_pos_y
20
21              if (self.pos_x > (self.start_pos_x+self.dist)):
22                  alpha_cur = math.acos((-self.pos_y + self.start_pos_y+self.R)/
    self.R)
23              else:
24                  alpha_cur = math.acos((self.pos_y - self.start_pos_y-self.R)/
    self.R)+math.pi
```

```
25
26                 if (alpha_cur >= self.alpha):
27                     fail = True
28                     allWay = True
29                     break
30
31                 self.cur_t += dt
32
33                 self.t_.append(self.cur_t)
34                 self.a_.append(self.cur_a)
35                 self.v_.append(self.cur_v)
36                 self.x_.append(self.pos_x)
37                 self.y_.append(self.pos_y)
38
39                 N = self.m *(a_c + self.g*math.cos(alpha_cur))
40
41                 if (N <= 0 or (alpha_cur == 0)):
42                     fail = True
43                     break
44
45             return allWay
```

Далее рассмотрим метод **ThirdPath()**, моделирующий третью часть пути:

```
1         def ThirdPath(self):
2             dt =1/self.toch
3             self.cur_a = -self.g
4             while self.pos_y >= self.start_pos_y:
5
6                 self.cur_vy += self.cur_a  * dt
7                 self.cur_v = math.sqrt(self.cur_vy**2 + self.cur_vx**2 )
8
9                 self.pos_x = max(self.pos_x +self.cur_vx*dt, 0)
10                self.pos_y = self.pos_y + self.cur_vy*dt +  self.cur_a*dt**2/2
11
12                #
13                if (self.pos_y <= self.UnderModel()):
14                    self.pos_y = self.UnderModel()
15                    self.t_.append(self.cur_t)
16                    self.a_.append(abs(self.cur_a))
```

```
                    self.v_.append(self.cur_v)

                    self.x_.append(self.pos_x)
                    self.y_.append(self.pos_y)
                    self.FifthPath()
                    break

                self.cur_t += dt
                time +=dt

                self.t_.append(self.cur_t)
                self.a_.append(abs(self.cur_a))
                self.v_.append(self.cur_v)
                self.x_.append(self.pos_x)
                self.y_.append(self.pos_y)
```

Далее рассмотрим метод **FourthPath()**, моделирующий четвертую часть пути:

```
        def FourthPath(self):
            dt =1/self.toch
            alpha_cur = math.acos((-self.pos_y + self.start_pos_y+self.R)/self.R)

            while (self.pos_x>(self.start_pos_x+self.dist)):

                a_c= self.cur_v**2/self.R
                a_t = -self.mu * (a_c + self.g*math.cos(alpha_cur)) + self.g*math.
    sin(alpha_cur)

                self.cur_ay = math.cos(alpha_cur)*a_c -math.sin(alpha_cur)*(a_t)
                self.cur_ax = -math.sin(alpha_cur)*a_c -math.cos(alpha_cur)*(a_t)
                self.cur_a = math.sqrt(self.cur_ax**2 + self.cur_ay**2)

                self.cur_vx += self.cur_ax*dt
                self.cur_vy += self.cur_ay*dt
                self.cur_v = math.sqrt(self.cur_vx**2 + self.cur_vy**2)

                self.pos_x = self.pos_x +self.cur_vx*dt + self.cur_ax*dt*dt/2
                self.pos_y = self.pos_y + self.cur_vy*dt + self.cur_ay*dt*dt/2,
    self.UnderModel()

```

```
                alpha_cur = math.acos((-self.pos_y + self.start_pos_y+self.R)/self
    .R)

                self.cur_t += dt
                time += dt

                self.t_.append(self.cur_t)
                self.a_.append(self.cur_a)
                self.v_.append(self.cur_v)
                self.x_.append(self.pos_x)
                self.y_.append(self.pos_y)
```

Далее рассмотрим метод **BinPoisk()**, находящий минимальное v_0:

```
        def BinPoisk(self):
        left = 2
        right = 1000
        while right - left > 0.00001:
            mid = (left+right)/2
            self.v_0 = mid
            if self.DO():
                right = mid
            else:
                left = mid
        self.v_0_ans = right
        return right
```

Вцелом тут классическая реализация алгоритма бинарного поиска. Проверяем среднее значение, если частица вылетела, значит рассматриваем середину как максимальное значение и ищем середину еще раз. Так, мы нашли значение v_0 с точностью до 0.00001.

4.2.4 Тело программы

Рассмотрим как вообще происходит отрисовка программы

```
while True:
    for event in pg.event.get():
        if event.type == pg.QUIT:
            pg.quit()
        if event.type == pg.MOUSEBUTTONDOWN:
                if input_boxV.collidepoint(event.pos):
                    active = not active
```

```
                elif input_boxA.collidepoint(event.pos):
                elif input_boxM.collidepoint(event.pos):
                elif input_boxR.collidepoint(event.pos):
                elif input_boxAl.collidepoint(event.pos):
                elif input_boxT.collidepoint(event.pos):
                else:
                    active = False
                color = color_active if active else color_inactive
                # ---
        if event.type == pg.KEYDOWN:
                if active:
                    if event.key == pg.K_RETURN:
                        print(text)
                        color = color_active
                        try:
                            u = text.find("U = ")
                            u = text[(u+4) :]
                            u = int(u)
                            print(u)
                        except:
                            text = 'U = '

                    elif event.key == pg.K_BACKSPACE:
                        text = text[:-1]
                    else:
                        text += event.unicode
                if activeR:
                if activeM:
                if activeA:
                if activeAl:
                if activeT:
    #model
    drawBase()
    el = pg.draw.circle(window,objectColor,(posStart,heigh),5)

    # Vo =
    txt_surface = font.render(textInputBoxV, True, color)
    width = max(100, txt_surface.get_width()+10)
```

```
    input_box.w = width
    window.blit(txt_surface, (input_box.x+5, input_box.y+5))
    pg.draw.rect(window, color, input_box, 2)
    #----
    # R =
    # mu =
    # alpha =
    # m =
    # Ao =
    #----
    # Run
    pg.draw.rect(window,  buttonActive, (350, 40, 65,33))
    txt_surface = font.render("RUN", True,  textColor)
    window.blit(txt_surface, (355, 45))

    # v_ans =
    Vans = font2.render(''someText ' ' + str(Model.v_0_ans), True,  textColor)
    window.blit(Vans, (50,550))

    window.blit(font.render(textT, True,  textColor), (600,200))
    window.blit(font.render(textV, True,  textColor), (600,250))
    window.blit(font.render(textA, True,  textColor), (600,300))

    Button.buttons.update()
    Button.buttons.draw(window)

    pg.display.update()
    clock.tick(FPS)
```

Пока программа открыта - работаем. Если Мы нажмем на крестик программы, то while закончится, это обрабатывает первый if внутри while.

Второй if внутри while обрабатывает считывание данных в текстовые поля

Третий if проверяет вводим ли мы что-то внутрь текстового поля или нет.

В следующих строках перерисовываем кнопки, текстовые поля и тд.

Далее рассмотрим метод **Model()**, вызывающийся при нажатии на кнопку RUN:

```
def model():
    posY = heigh
    posx = posStart

    Model.v_0 = v
    Model.a_0 = a
    Model.DO()

    for i in range (0, len(Model.x_)):
        drawBase()

        forX = posx+(Model.x_[i]-posx)*mnozh
        forY = posY-(Model.y_[i]-posY)*mnozh
        if (forX < -5 or forY < -5):
            continue
        pg.draw.circle(window,objectColor,(forX, posY-(Model.y_[i]-posY)*mnozh),5)

        # inputTexts
        txt_surface = font.render(textInputBoxV, True, color_inactive)
        width = max(100, txt_surface.get_width()+10)
        input_box.w = width
        window.blit(txt_surface, (input_box.x+5, input_box.y+5))
        pg.draw.rect(window, color, input_box, 2)

        #---

        Uans = font2.render('someText ' + str(Model.v_0_ans), True,  textColor)
        window.blit(Uans, (50,550))

        textT = str(Model.t_[i])
        point = textT.find('.')
        textT = 't: ' + textT[:point+3]+ ' c'
        textV = '|v|: ' + f'{Model.v_[i]:.2f}' + '   /  '
        textA = '|a|: ' + f'{Model.a_[i]:.2f}'+ '   /  ^2'

        window.blit(font.render(textT, True,  textColor), (600,200))
        window.blit(font.render(textV, True,  textColor), (600,250))
        window.blit(font.render(textA, True, textColor), (600,300))
```

```
39
40          pg.display.update()
```

В строке 7 запускаем модельку с полученным значением v_0 и a.
Далее проходимся по полученному массиву и в строках 12-16 нормируем полученные положения электрона к длине и высоте нарисованного конденсатора.
Далее отрисовываем поля с временем и скоростью

4.3 Результат

С полным кодом вы можете ознакомиться и скачать его тут
А пример работы тут

Исследование

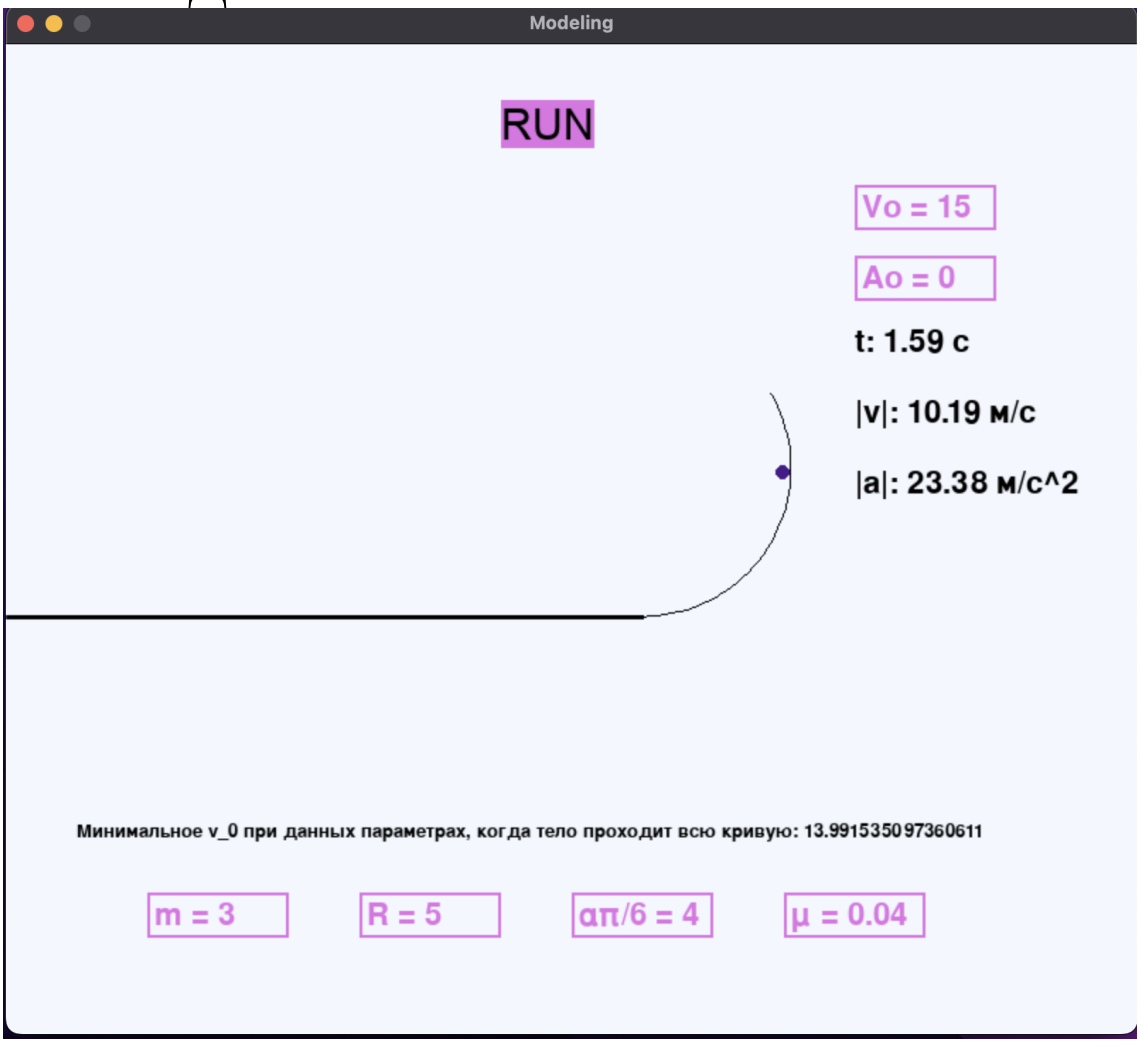

Рис. 5.1: Работа программы при заданных условием параметрах

Для начального ускорения $a = 0$ минимальное начальное значение v при которой тело прройдет всю длину дуги:

$$v_0 = 13.991535097360611$$

Что касательно траектории тела после отрыва, так это баллистическая траектория, ведь движение явялется равномерным по x и равноускоренным по y

Выводы

В ходе работы мной была исследована математическая модель движения тела вверх по дуге окружности с начальной скоростью. Были выведены формулы для ускорения,времени, изменения скорости , положения на каждом участке пути(1 часть,2 часть,3 часть,4 часть). С помощью этого удалось выполнить моделирование на языке программирования python, с результатами можно ознакомиться тут. Благодаря ему нами стало известно значение минимального v_0, при котором тело пройдет всю длину дуги($v_{0=}$13.991535097360611 м/с). Также мы увидели, что траектория движения тела после отрыва похожа на баллистическую, причиной этому является равномерное движение по x и равноускоренное по y.