Моделирование физического процесса "Частица в конденсаторе"

Группа M3203
Кравченкова Елизавета

Преподаватель
Хуснутдинова Наира Рустемовна

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

11 декабря 2023 г.

Оглавление

Задание

Электрон влетает в цилиндрический конденсатор с начальной скоростью V, посередине между обкладками, параллельно образующим цилиндра. При какой минимальной разности потенциалов, приложенной к обкладкам, электрон не успеет вылететь из конденсатора. Краевыми эффектами пренебречь.

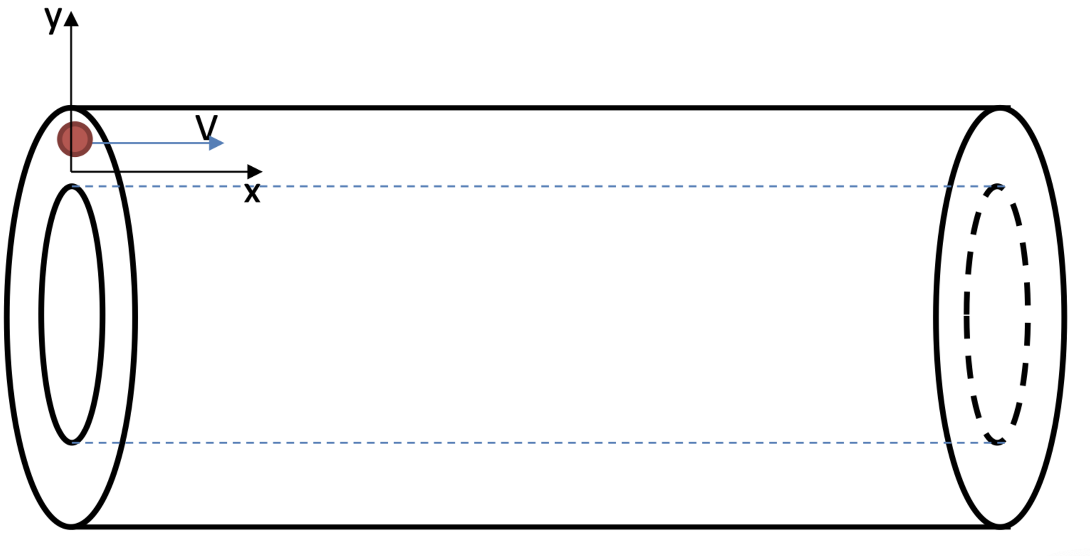

Рис. 1.1: Иллюстрация к задаче

Построить графики зависимости y(x), Vy(t), ay(t), y(t). Координатные оси направлены как показано на рисунке. Рассчитать время полета t и конечную скорость электрона Vкон. Данные по размерам конденсатора и скорости электрона взять из таблицы. Номер варианта соответствует номеру по списку группы.
Вариант 9:
Внутренний радиус r = 5 см
Внешний радиус R = 11 см
Начальная скорость V = 4*10**6 м/с
Длина конденсатора L = 19 см

1.1 Объект исследования

Исследование движения электрона в цилиндрическом конденсаторе

1.2 Метод экспериментального исследования

Имитация физической модели с помощью языка программирования

Цели и Задачи

Цель: Выполнить численное моделирование движения электрона в конденсаторе с заданной начальной скоростью. Построить графики и найти требуемые величины.

Задачи:

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

Теория и рабочие формулы

Крайний случай, когда частица не смогла вылететь, это если она попала в точку B. Рассмотрим действующие на частицу силы.

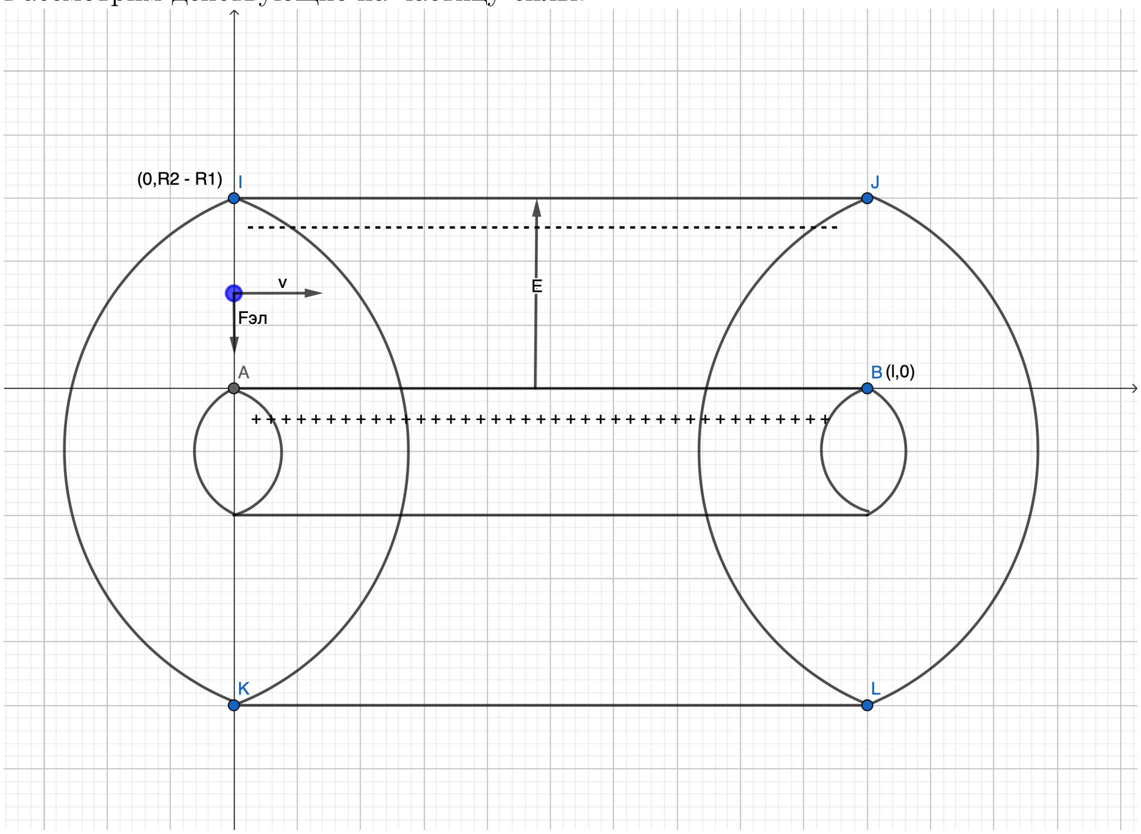

Рис. 3.1: Силы

$$\vec{F_{эл}} = e\vec{E}$$

По второму закону Ньютона:

$$\vec{F_{эл}} = m\vec{a}$$

В проекциях на Oy:

$$-F_{эл} = -ma_y$$

$$F_{эл} = ma_y$$

$$eE = -ma_y$$

$$a_y = -\frac{eE}{m} \tag{1}$$

В цилиндрическом конденсаторе:

$$E(r) = \frac{q}{2\pi\varepsilon\varepsilon_0 rl}$$

$$\Delta\phi = U = \frac{q}{2\pi\varepsilon\varepsilon_0 l}\ln\frac{R_2}{R_1}$$

Откуда можно получить соотношение:

$$U = E(r) r \ln\frac{R_2}{R_1}$$

Подставим это в (1)

$$a_y = -\frac{eU}{rm\ln\frac{R_2}{R_1}} \tag{2}$$

Таким образом меняя начальное значение U, меняется ускорение и как следствие все остальное. Так как ускорение переменное, то стандартные формулы кинематики не работают. В нашем моделировании Будут рассматриваться очень малые промежутки времени, в рамках которых ускорение постоянно, тогда:

$$\Delta v_y = a_y \Delta t \tag{3}$$

$$\Delta y = -v_y - \frac{a_y \Delta t^2}{2} \tag{4}$$

По оси y тело движется с ускорением, по оси x же без:

$$v_x = const$$

$$v_x t = l$$

$$t = \frac{l}{v_x} \tag{5}$$

Таким образом всегда можно найти время, зная расстояние, которое пролетела частица.

Ход работы

4.1 Поготовка

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

```
pip install matplotlib
pip install pygame
```

4.2 Основная часть

4.2.1 Задание констант

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

```
pg.init()

WIDTH, HEIGHT = 800, 700
FPS = 60

window = pg.display.set_mode((WIDTH, HEIGHT))
pg.display.set_caption('Modeling')
window.fill(pg.Color('white'))
clock = pg.time.Clock()

#-------
toch = 1000
e = 1.6 * 10**(-19)
m = 9.1 * 10**(-31)
v = 4.5 * 10**6 # m/c
l = 0.19 #m
R1 = 0.05 #m
R2 = 0.11 #m
```

4.2.2 Создание модели

Создадим класс Model, который будет хранить текущее положение электрона, массивы позиций по x и y, а также другие массивы для построения графиков. Также этот класс при создании экземляра с помощью алгоритма бинарного поиска подбирает такое значение минимальное значение U, при котором частица останется в конденсаторе.(но об этом позже). Еще этот класс имеет метод DO, именно он выполняет всю логику по перемещению электрона(об этом тоже позже). Вообщем наш класс обладает следующей структурой:

```
class Model:
    U = 0
    U_ans = 0
    x_= []
    y_= []
    t_ = []
    Vy = []
    Ay = []
    t_ans =0
    v_ans = 0

    def __init__(self):
        U_ans = self.BinPoisk()

    def clean(self)
    def DO(self)
    def GetGrap(self)
    def BinPoisk(self)
```

4.2.3 Реализация основных функций физической модели

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

```
    def DO(self):
        self.clean()
        a_koef = self.e*self.U/(self.m*math.log(self.R2/self.R1))
        pos = self.d/2
        r = pos + self.R1
        a = a_koef/r
```

```
        vy = 0
        t = 0
        dt = self.l/(self.v*self.toch)
        for q in range(0,self.toch):
            self.t_.append(t)
            self.Ay.append(a)
            self.Vy.append(vy)

            self.x_.append(self.v*t)
            self.y_.append(pos)

            if (pos == 0):
                break

            pos = max (pos - vy*dt -  a*dt**2/2, 0)
            vy = vy if pos == 0 else vy + a*dt
            r = pos + self.R1
            a = a if pos == 0 else a_koef/r

            t += dt
            self.t_ans = t
            if (t > (self.l/self.v)):
                break
        self.v_ans = math.sqrt(self.v**2 + vy**2)
        self.t_ans = t
        return pos == 0
```

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

```
        def BinPoisk(self):
        left = 2
        right = 100
        while right - left > 0.00001:
            mid = (left+right)/2
            self.U = mid
            if self.Do():
                right = mid
            else:
                left = mid
        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 U с точностью до 0.00001.
Далее рассмотрим метод **GetGrap()**, строящий графики:

```
        def GetGrap(self):
        plt.subplot (2, 2, 1)
        plt.plot(self.x_, self.y_)
        plt.axis((0, self.l, 0, self.R2-self.R1))
        plt.title("  (x)")

        plt.subplot (2, 2, 2)
        plt.plot(self.t_, self.Vy)
        plt.title("v_y(t)")

        plt.subplot (2, 2, 3)
        plt.plot(self.t_, self.Ay)
        plt.title("a_y(t)")

        plt.subplot (2, 2, 4)
        plt.plot(self.t_, self.y_)
        plt.title("y(t)")
        #
        plt.show()
```

На этом моменте должно стать понятно, почему мы храним массивы наших посчитанных значений.

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_box.collidepoint(event.pos):
                    active = not active
                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
    drawBase()
    txt_surface = font.render(text, 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)
    pg.draw.rect(window,  pg.Color('green'), (350, 40, 65,33))
    txt_surface = font.render("RUN", True,  pg.Color('black'))
    window.blit(txt_surface, (355, 45))
    Uans = font2.render('min U: ' + str(Model.U_ans), True,  pg.Color('black'))
    window.blit(Uans, (50,550))

    window.blit(font.render(textT, True,  pg.Color('black')), (600,200))
    window.blit(font.render(textV, True,  pg.Color('black')), (600,250))

    buttons.update()
    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 строках 28-46 перерисовываем кнопки, текстовые поля и тд.

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

```python

def model():
    posY = 150
    posx = 100
    lW = 400
    hW = 50

    Model.U = u
    flag = Model.DO()
    x = Model.x_
    y = Model.y_
    textT = textT[:point+3]+ ' sec'
    textV = f'{Model.v_ans:.2f}' + ' m/c'

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

        pg.draw.circle(window,pg.Color('blue'),(posx+x[i]*lW/Model.l, posY  + hW - 2*y[i]*hW/Model.d),5)

        window.blit(font.render(textT, True,  pg.Color('black')), (600,200))
        window.blit(font.render(textV, True,  pg.Color('black')), (600,250))
        pg.display.update()

    col = (pg.Color(255, 105, 105)) # red
    if flag :
        col = pg.Color(193, 242, 176) # green

    while True:
        for event in pg.event.get():
            if event.type == pg.QUIT:
```

```
31                  pg.quit()
32                  quit()
33          if event.type == pg.MOUSEBUTTONDOWN:
34                  if rec.collidepoint(event.pos):
35                      Model.GetGrap()
36                      break
37 
38          drawBase(col)
39 
40          window.blit(font.render(textT, True,  pg.Color('black')), (600,200))
41          window.blit(font.render(textV, True,  pg.Color('black')), (600,250))
42 
43          pg.display.update()
```

В строках 8-13 запускаем модельку с полученным значением U и берем из нее нужные данные. Далее проходимся по полученному массиву и в 18 строке нормируем полученные положения электрона к длине и высоте нарисованного конденсатора.
Далее отрисовываем поля с временем и скоростью
В 24-26 строке меняем цвет фона в зависимости от результата работы модельки.
Далее запускаем while, который ждет, когда человек нажмет кнопку графики. Когда он ее нажимает(37 строка), вызывается метод модели для получения графиков.

4.3 Результат

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

Исследование

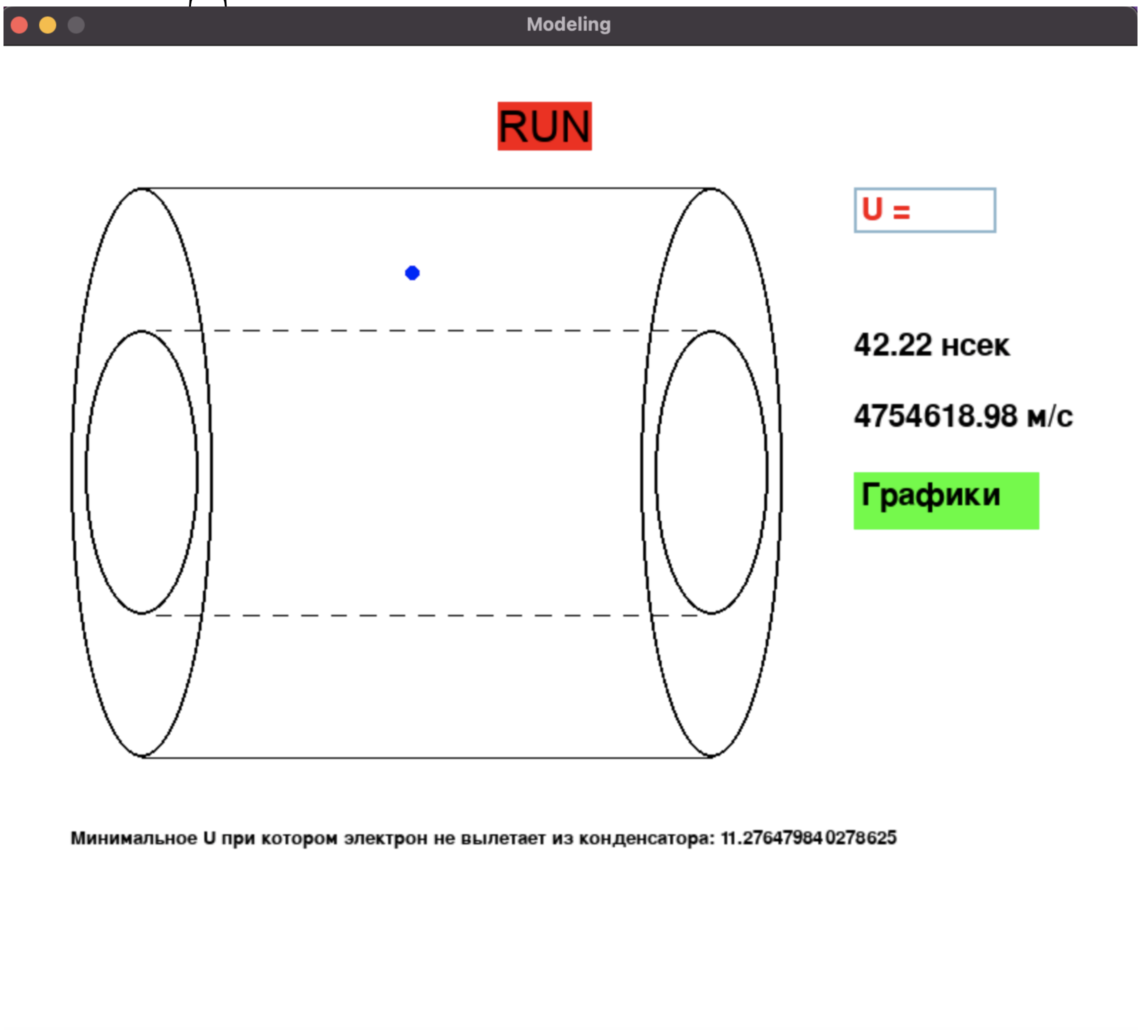

Рис. 5.1: Работа программы при минимальном U

Полученные значения:

$$U_{\text{мин}} = 11.276479840278625\text{В}$$

$$t = 42.22\text{нсек}$$

$$v_{\text{кн}} = 4754618.98\text{м/с}$$

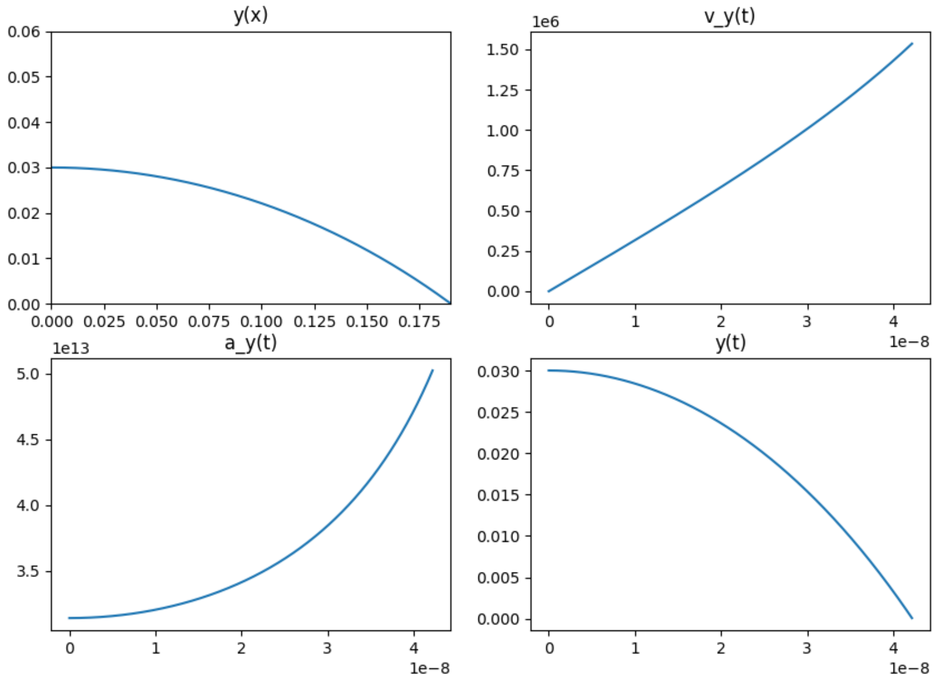

Рис. 5.2: Графики

На графиках можем увидеть, что траектория движения электрона похожа на баллистическую, причиной этому является равномерное движение по x и ускоренное по y. Рост ускорения нелинейный, причина этого, что r(расстояние до центра меньшего цилиндра) также уменьшается нелинейно, а по баллистике. Изменение скорости более сбалансированное, хотя если приглядется, можно увидеть, что это все еще не идеальная прямая. График y(t) почти полностью повторяет y(x) так как движение по x равномерное($x = v_x t$), а значит график y(x) это лишь расширение y(t) в v_x раз.

Выводы

В ходе работы мной была исследована математическая модель движения электрона в цилиндрическом конденсаторе с начальной скоростью. Была выведена формула для ускорения (2),времени (5), изменения скорости (3), положения (4). С помощью этого удалось выполнить моделирование на языке программирования python, с результатами можно ознакомиться тут. Благодаря ему нами стали известны значения минимального U, при котором электрон не вылетит из конденсатора($U_{\text{мин}} = 11.276479840278625$ В), время его полета ($t = 42.22$нсек), а также конечная скорость ($v_{\text{кн}} = 4754618.98$м/с). Также нами получены графики (смотри тут). На них мы увидели, что траектория движения электрона похожа на баллистическую, причиной этому является равномерное движение по x и ускоренное по y. Рост ускорения нелинейный, причина этого, что r (расстояние до центра меньшего цилиндра) также уменьшается нелинейно, а по баллистике. Изменение скорости более сбалансированное, хотя если приглядется, можно увидеть, что это все еще не идеальная прямая. График y(t) почти полностью повторяет y(x) так как движение по x равномерное($x = v_x t$), а значит график y(x) это лишь расширение y(t) в v_x раз.