Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по лабораторной работе №2**

По дисциплине: «Компьютерные системы моделирования»

Тема: «Аналитическое моделирование. Физическое компьютерное моделирование»

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС - 22.06  Иванов А.М. |
| Проверила:  старший преподаватель  кафедры ПО  Корнеева Е.И. |

Тверь 2025

**Оглавление**

[**Постановка задачи для варианта.** 3](#_Toc188991895)

[**Краткий конспект теории к задаче из текста работы.** 4](#_Toc188991896)

[**Математическая модель задачи.** 6](#_Toc188991897)

[**Отчет по реализации в табличном редакторе Microsoft.** 7](#_Toc188991898)

[**Отчет о реализации на Python** 12](#_Toc188991899)

[**Тестирование** 26](#_Toc188991900)

[**Выводы по результатам решения задачи.** 32](#_Toc188991901)

# **Постановка задачи для варианта.**

Необходимо описать и реализовать математическую модель, продумать архитектуру, оформить визуальный интерфейс и сделать покрытие тестами, также реализовать трассировку значений важных для задачи переменных.

Необходимо изучить влияние вязкости среды на характер движения.

Шар, сделанный из олова радиуса r = 0,15 м падает в мазуте, встречая

силу сопротивления, пропорциональную скорости. Найти изменение

скорости и высоты падения при изменении времени. Построить графики

зависимости скорости и высоты от времени.

Реализовать решение в 2 программах:

Поиск решения задачи в MS Excel (VBA).

C помощью python посчитать и вывести таблицу с ответами. Для графической части выбрать фреймворк pyQt6, pySide6 и аналоги. TKinter для задачи не использовать. Графики нарисовать с помощью matplotlib.

# **Краткий конспект теории к задаче из текста работы.**

**Основные физические законы и понятия**

При падении шара в вязкой среде, например, мазуте, на него действуют несколько сил:

1. **Сила тяжести** *Fg*​, которая направлена вниз и пропорциональна массе объекта: *Fg*​=*mg* где *m* — масса шара, *g* — ускорение свободного падения.
2. **Сила выталкивания (архимедова сила)** *Fb*​, которая направлена вверх и пропорциональна объему погруженной части объекта и плотности жидкости: *Fb*​=*V\*ρmed*​\**g* где *V* — объем шара, *ρmed*​ — плотность среды.
3. **Сила сопротивления среды** *Fd*​, которая направлена против движения и пропорциональна скорости шара: *Fd*​=−*kv* где *k* — коэффициент сопротивления, зависящий от вязкости среды *η*, радиуса шара *r* и других геометрических параметров: *k*=6*πηr*

Для шара, падающего в вязкой среде, динамика его движения описывается вторым законом Ньютона: *mdtdv*​=*Fg*​−*Fb*​−*Fd*​.

Подставляя выражения для сил, получаем: *mdtdv*​=(*ρobj*​−*ρmed*​)*Vg*−*kv*

**Терминальная скорость**

Терминальная скорость *v*terminal​ достигается, когда суммарная сила становится равной нулю:

(*ρobj*​−*ρmed*​)*Vg*=*kv*terminal​

*v*terminal​=*k*(*ρobj*​−*ρmed*​)*Vg*​

**Решение дифференциального уравнения**

Уравнение движения можно решить аналитически, если начальные условия заданы как *v*(0)=0. В результате решения получаем выражения для скорости *v*(*t*) и высоты *h*(*t*):

*v*(*t*)=*v*terminal​(1−*e*−*mkt*​)

*h*(*t*)=*v*terminal​*t*−*kmv*terminal​​(1−*e*−*mkt*​)

# **Математическая модель задачи.**

**Инициализация модели**

Модель начинается с инициализации параметров шара и среды:

* *r*: радиус шара (м)
* *ρobj*​: плотность материала шара (кг/м³)
* *ρmed*​: плотность среды (кг/м³)
* *g*: ускорение свободного падения (м/с²)
* *tmax*​: максимальное время моделирования (с)
* *dt*: шаг времени (с)

**Предварительные расчеты**

Перед началом решения необходимо выполнить предварительные расчеты:

* Объем шара *V*: *V*=34​*πr*3
* Эффективная масса шара *m*: *m*=(*ρobj*​−*ρmed*​)*V*
* Временной массив *t*: *t*=[0,*dt*,2*dt*,...,*tmax*​]

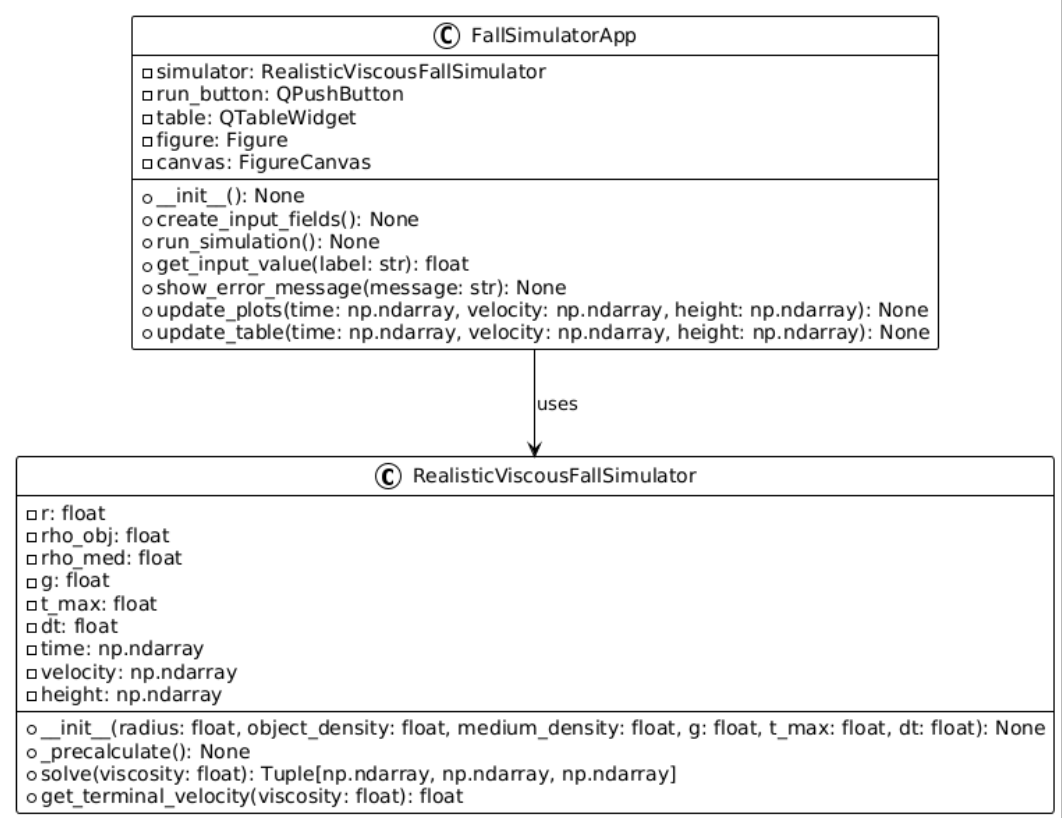
**Решение задачи**

Решение задачи выполняется через следующие шаги:

1. Вычисление коэффициента сопротивления *k*: *k*=6*πηr*
2. Вычисление терминальной скорости *v*terminal​: *v*terminal​=*kmg*​
3. Вычисление скорости *v*(*t*) и высоты *h*(*t*) в каждый момент времени: *v*(*t*)=*v*terminal​(1−*e*−*mkt*​) *h*(*t*)=*v*terminal​*t*−*kmv*terminal​​(1−*e*−*mkt*​)

# **Отчет по реализации в табличном редакторе Microsoft.**

**Схема программы:**



1. Ввод параметров:

Пользователь задает параметры модели (радиус, плотность материала шара, плотность среды, вязкость, ускорение свободного падения, максимальное время и шаг времени) в соответствующих ячейках на листе.

2. Кнопка запуска симуляции:

По нажатию кнопки вызывается макрос `RunSimulation`, который считывает значения из ячеек, выполняет расчеты и выводит результаты в таблицу.

3. Расчеты:

1. Вычисляются объем и масса шара.

2. Рассчитывается коэффициент сопротивления и терминальная скорость.

3. В цикле вычисляются значения скорости и высоты для каждого момента времени.

4. Вывод результатов:

1. Результаты расчетов выводятся в таблицу на листе.

2. Графики зависимости скорости и высоты от времени строятся на основе данных таблицы.

**Листинг кода с пояснением**

**Листинг 1: Макрос VBA для выполнения симуляции**

Private Sub CommandButton1\_Click()

RunSimulation

End Sub

Sub RunSimulation()

Dim radius As Double

Dim objectDensity As Double

Dim mediumDensity As Double

Dim viscosity As Double

Dim g As Double

Dim tMax As Double

Dim dt As Double

' Чтение параметров из ячеек

radius = Range("B1").Value

objectDensity = Range("B2").Value

mediumDensity = Range("B3").Value

viscosity = Range("B4").Value

g = Range("B5").Value

tMax = Range("B6").Value

dt = Range("B7").Value

Dim volume As Double

Dim mass As Double

Dim k As Double

Dim vTerminal As Double

Dim time() As Double

Dim velocity() As Double

Dim height() As Double

Dim i As Integer

Dim t As Double

' Предварительные расчеты

volume = (4 / 3) \* Application.WorksheetFunction.Pi() \* radius ^ 3

mass = (objectDensity - mediumDensity) \* volume

k = 6 \* Application.WorksheetFunction.Pi() \* viscosity \* radius

vTerminal = (mass \* g) / k

' Инициализация массивов для времени, скорости и высоты

ReDim time(0 To tMax / dt)

ReDim velocity(0 To tMax / dt)

ReDim height(0 To tMax / dt)

' Цикл расчета значений скорости и высоты

For i = 0 To tMax / dt

t = i \* dt

time(i) = t

velocity(i) = vTerminal \* (1 - Exp(-k \* t / mass))

height(i) = vTerminal \* t - (mass \* vTerminal / k) \* (1 - Exp(-k \* t / mass))

Next i

' Вывод результатов в таблицу

For i = 0 To UBound(time)

Cells(i + 2, 4).Value = time(i)

Cells(i + 2, 5).Value = velocity(i)

Cells(i + 2, 6).Value = height(i)

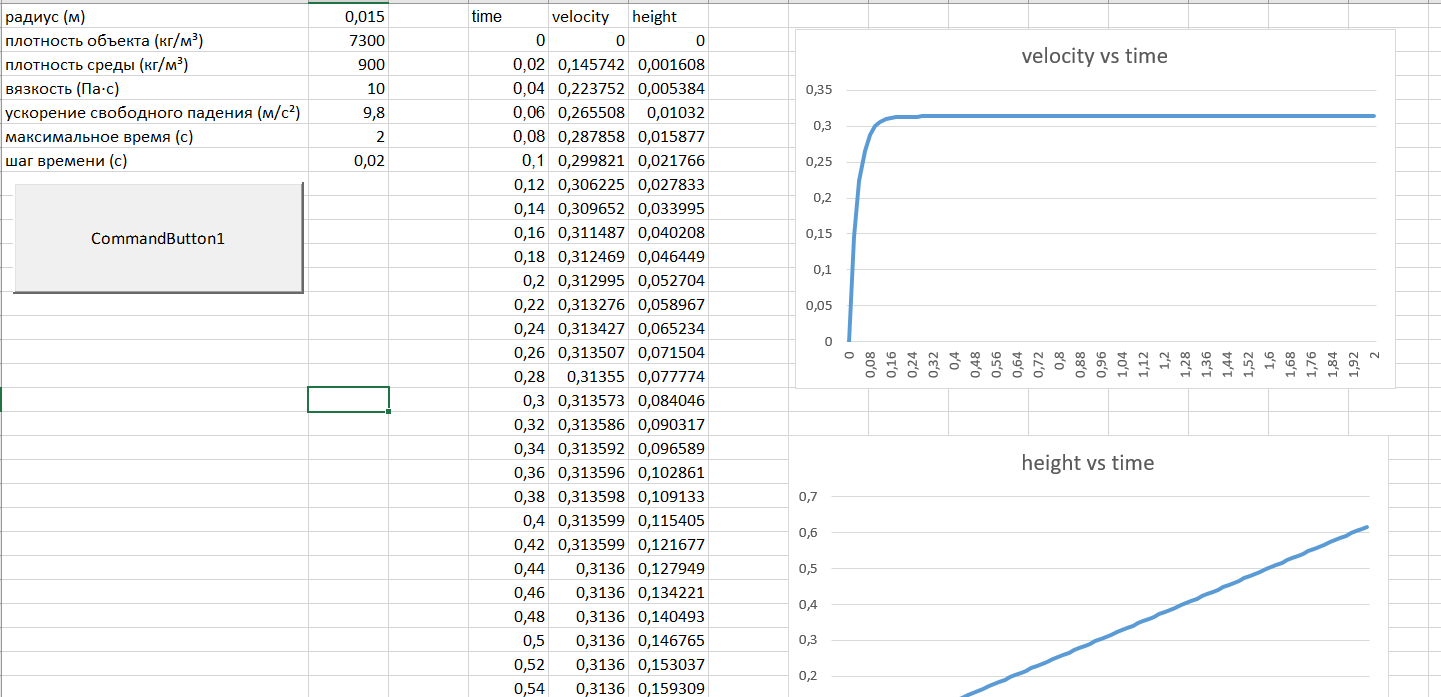
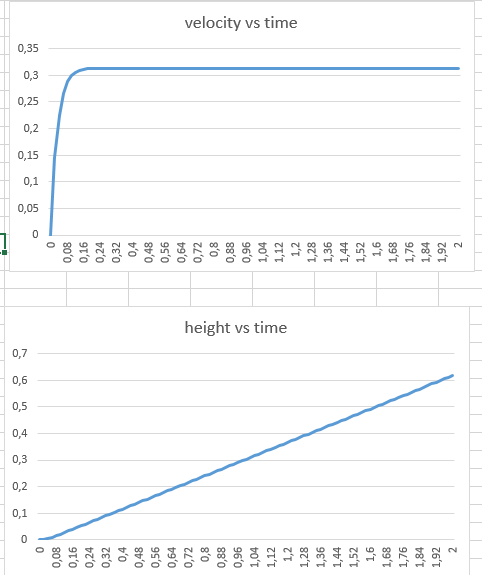
Next i

End Sub

Пояснение:

1. CommandButton1\_Click: При нажатии на кнопку вызывается процедура `RunSimulation`.
2. Считывает значения параметров из ячеек B1-B7.
3. Выполняет предварительные расчеты: объем, масса, коэффициент сопротивления и терминальная скорость.
4. Инициализирует массивы для времени, скорости и высоты.
5. В цикле рассчитывает значения скорости и высоты для каждого момента времени и записывает их в таблицу.

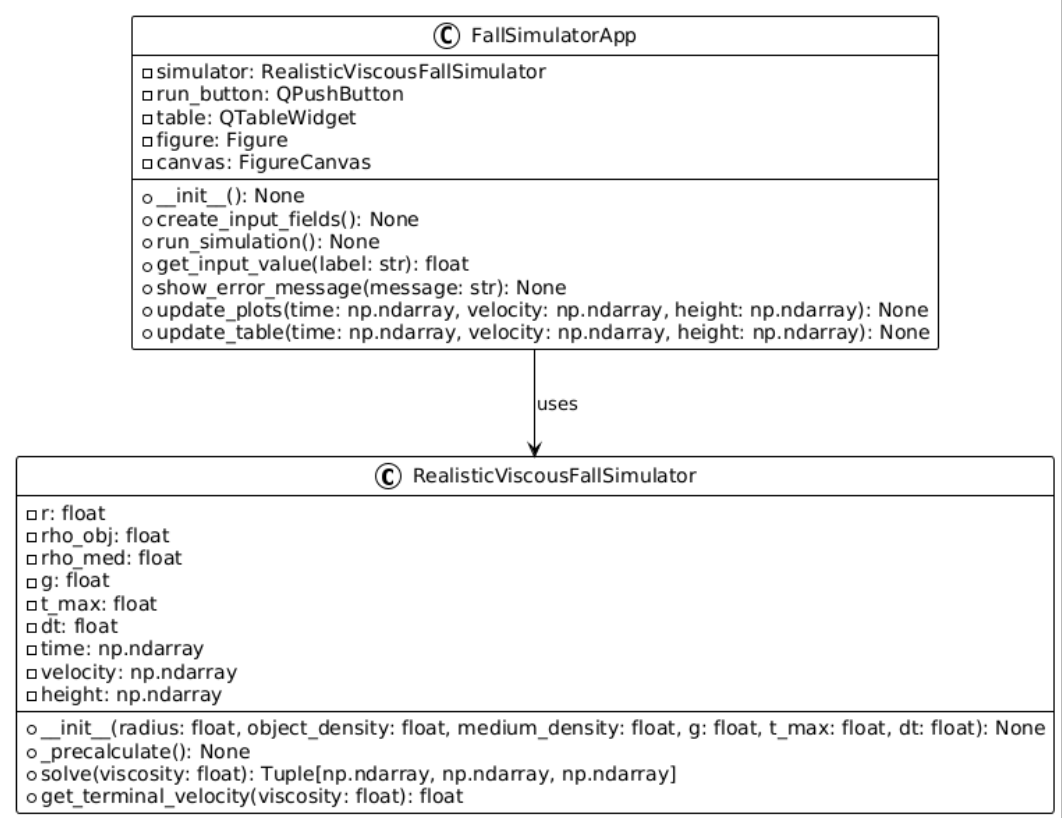
**Скриншоты выполнения работы**

**  
**

# **Отчет о реализации на Python**

**Схема программы и псевдокод**

**Схема программы**



1. Класс RealisticViscousFallSimulator:

1. Инициализация параметров.

2. Предварительные расчеты (объем, масса).

3. Метод для решения задачи падения с учетом вязкости.

4. Метод для получения терминальной скорости.

2. Главный блок:

1. Создание экземпляра класса RealisticViscousFallSimulator.

2. Решение задачи для разных значений вязкости.

3. Построение графиков зависимости скорости и высоты от времени.

3. Графический интерфейс (FallSimulatorApp):

1. Создание окна приложения.

2. Создание полей ввода для параметров симуляции.

3. Запуск симуляции по кнопке.

4. Обновление таблицы и графиков после выполнения симуляции.

**Псевдокод**

class RealisticViscousFallSimulator:

def \_\_init\_\_(self, radius, object\_density, medium\_density, g, t\_max, dt):

# Инициализация параметров модели

def \_precalculate(self):

# Предварительные расчеты (объем, масса)

def solve(self, viscosity):

# Решение задачи падения с учетом вязкости

def get\_terminal\_velocity(self, viscosity):

# Возвращает терминальную скорость

if \_\_name\_\_ == '\_\_main\_\_':

# Создание экземпляра класса

simulator = RealisticViscousFallSimulator(...)

# Решение задачи для разных значений вязкости

for viscosity in viscosities:

time, velocity, height = simulator.solve(viscosity)

# Построение графиков

# Графический интерфейс

class FallSimulatorApp(QMainWindow):

def \_\_init\_\_(self):

# Инициализация интерфейса

def create\_input\_fields(self):

# Создание полей ввода

def run\_simulation(self):

# Запуск симуляции

def update\_plots(self, time, velocity, height):

# Обновление графиков

def update\_table(self, time, velocity, height):

# Обновление таблицы

if \_\_name\_\_ == "\_\_main\_\_":

# Создание и запуск приложения

**Листинг кода с пояснением**

**Листинг 1: Класс RealisticViscousFallSimulator**

import numpy as np

*class* RealisticViscousFallSimulator:

*def* \_\_init\_\_(*self*, *radius*: float, *object\_density*: float, *medium\_density*: float,

*g*: float = 9.81, *t\_max*: float = 20, *dt*: float = 0.1) -> None:

        """

        Инициализация модели с учетом плотности среды.

        :param radius: радиус шара (м)

        :param object\_density: плотность материала шара (кг/м³)

        :param medium\_density: плотность среды (кг/м³)

        :param g: ускорение свободного падения (м/с²)

        :param t\_max: максимальное время моделирования (с)

        :param dt: шаг времени (с)

        """

        if *radius* < 0 or *object\_density* < 0 or *medium\_density* < 0 or *g* < 0 or *t\_max* < 0 or *dt* <= 0:

            raise ValueError("Все параметры должны быть неотрицательными, а dt должен быть положительным.")

*self*.r = *radius*

*self*.rho\_obj = *object\_density*

*self*.rho\_med = *medium\_density*

*self*.g = *g*

*self*.t\_max = *t\_max*

*self*.dt = *dt*

*self*.time = None

*self*.velocity = None

*self*.height = None

*self*.\_precalculate()

*def* \_precalculate(*self*) -> None:

        """Предварительные расчеты."""

*self*.volume = (4 / 3) \* np.pi \* *self*.r\*\*3

*self*.mass = (*self*.rho\_obj - *self*.rho\_med) \* *self*.volume

*self*.time = np.arange(0, *self*.t\_max + *self*.dt, *self*.dt)

*def* solve(*self*, *viscosity*: float) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:

        """

        Решение задачи падения с учетом вязкости.

        :param viscosity: вязкость среды (Па·с)

        :return: массивы времени, скорости и высоты

        """

        if *viscosity* < 0:

            raise ValueError("Вязкость среды должна быть положительной.")

        k = 6 \* np.pi \* *viscosity* \* *self*.r

        print(*f*"Коэффициент сопротивления (k): {k}")  # трейс k

        v\_terminal = (*self*.mass \* *self*.g) / k

        print(*f*"Терминальная скорость (v\_terminal): {v\_terminal}")  # трейс v\_terminal

        t = *self*.time

*self*.velocity = v\_terminal \* (1 - np.exp(-k \* t / *self*.mass))

        print(*f*"Скорость (velocity): {*self*.velocity}")  # трейс velocity

*self*.height = v\_terminal \* t - (*self*.mass \* v\_terminal / k) \* (1 - np.exp(-k \* t / *self*.mass))

        print(*f*"Высота (height): {*self*.height}")  # трейс height

        return *self*.time, *self*.velocity, *self*.height

*def* get\_terminal\_velocity(*self*, *viscosity*: float) -> float:

        """

        Возвращает терминальную скорость.

        :param viscosity: вязкость среды (Па·с)

        :return: терминальная скорость (м/с)

        """

        k = 6 \* np.pi \* *viscosity* \* *self*.r

        return (*self*.mass \* *self*.g) / k

Пояснение:

* 1. Конструктор инициализирует параметры модели и вызывает метод \_precalculate для предварительных расчетов. Выполняется проверка корректности ввода.
  2. Метод \_precalculate вычисляет объем и массу шара.
  3. Метод solve решает задачу падения с учетом вязкости среды и возвращает массивы времени, скорости и высоты.
  4. Метод get\_terminal\_velocity возвращает терминальную скорость для заданной вязкости.

Листинг 2: Главный блок программы

if \_\_name\_\_ == '\_\_main\_\_':

params = {

'radius': 0.015, # 1.5 см

'object\_density': 7300, # олово

'medium\_density': 900, # мазут

't\_max': 10,

'dt': 0.1

}

simulator = RealisticViscousFallSimulator(\*\*params)

viscosities = [1, 10, 100]

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

for eta in viscosities:

time, velocity, height = simulator.solve(eta)

ax1.plot(time, velocity, label=f'η = {eta} Па·с')

ax2.plot(time, height, label=f'η = {eta} Па·с')

ax1.set\_ylim(0, 5)

ax2.set\_ylim(0, 30)

ax1.set\_title('Сравнение скоростей для разных вязкостей')

ax1.set\_ylabel('Скорость (м/с)')

ax1.legend()

ax1.grid(True)

ax2.set\_title('Сравнение высот для разных вязкостей')

ax2.set\_xlabel('Время (с)')

ax2.set\_ylabel('Высота (м)')

ax2.legend()

ax2.grid(True)

plt.tight\_layout()

plt.show()

**Пояснение**:

1. Создается экземпляр класса RealisticViscousFallSimulato с заданными параметрами.

2. Для нескольких значений вязкости вызывается метод solve, результаты которого используются для построения графиков скорости и высоты.

**Листинг 3: Графический интерфейс (FallSimulatorApp)**

class FallSimulatorApp(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Simulate Viscous Fall")

self.setGeometry(100, 100, 1200, 800)

self.simulator = None

main\_widget = QWidget()

self.setCentralWidget(main\_widget)

main\_layout = QHBoxLayout()

main\_widget.setLayout(main\_layout)

left\_layout = QVBoxLayout()

main\_layout.addLayout(left\_layout, 1)

self.input\_layout = QGridLayout()

left\_layout.addLayout(self.input\_layout)

self.create\_input\_fields()

self.run\_button = QPushButton("Run Simulation")

self.run\_button.clicked.connect(self.run\_simulation)

left\_layout.addWidget(self.run\_button)

self.table = QTableWidget()

self.table.setColumnCount(3)

self.table.setHorizontalHeaderLabels(["Time (s)", "Velocity (m/s)", "Height (m)"])

left\_layout.addWidget(self.table)

right\_layout = QVBoxLayout()

main\_layout.addLayout(right\_layout, 3)

self.figure = Figure()

self.canvas = FigureCanvas(self.figure)

right\_layout.addWidget(self.canvas)

def create\_input\_fields(self):

"""

Создаем поля ввода для параметров симуляции.

"""

labels = [

"Radius (m):", "Object Density (kg/m³):", "Medium Density (kg/m³):",

"Viscosity (Pa·s):", "Gravity (m/s²):", "Max Time (s):", "Time Step (s):"

]

self.inputs = {}

defaults = [0.01, 1000, 998, 0.001, 9.81, 20, 0.1]

for i, (label, default) in enumerate(zip(labels, defaults)):

self.input\_layout.addWidget(QLabel(label), i, 0)

line\_edit = QLineEdit(str(default))

self.input\_layout.addWidget(line\_edit, i, 1)

self.inputs[label] = line\_edit

def run\_simulation(self):

"""

Запуск симуляции.

"""

try:

radius = self.get\_input\_value("Radius (m):")

object\_density = self.get\_input\_value("Object Density (kg/m³):")

medium\_density = self.get\_input\_value("Medium Density (kg/m³):")

viscosity = self.get\_input\_value("Viscosity (Pa·s):")

g = self.get\_input\_value("Gravity (m/s²):")

t\_max = self.get\_input\_value("Max Time (s):")

dt = self.get\_input\_value("Time Step (s):")

self.simulator = RealisticViscousFallSimulator(

radius, object\_density, medium\_density, g, t\_max, dt

)

time, velocity, height = self.simulator.solve(viscosity)

self.update\_plots(time, velocity, height)

self.update\_table(time, velocity, height)

except ValueError as e:

self.show\_error\_message(str(e))

def get\_input\_value(self, label):

"""

Получает и проверяет значение ввода.

"""

try:

return float(self.inputs[label].text())

except ValueError:

raise ValueError(f"Неверное значение для '{label}'")

def show\_error\_message(self, message):

"""

Показывает сообщение об ошибке.

"""

msg\_box = QMessageBox(self)

msg\_box.setIcon(QMessageBox.Critical)

msg\_box.setWindowTitle("Ошибка ввода")

msg\_box.setText(message)

msg\_box.exec()

def update\_plots(self, time, velocity, height):

"""

Обновление графиков.

"""

self.figure.clear()

ax1 = self.figure.add\_subplot(211)

ax2 = self.figure.add\_subplot(212)

ax1.plot(time, velocity, label="Velocity", color="blue")

ax1.set\_title("Velocity vs Time")

ax1.set\_xlabel("Time (s)")

ax1.set\_ylabel("Velocity (m/s)")

ax1.legend()

ax1.grid()

ax2.plot(time, height, label="Height", color="green")

ax2.set\_title("Height vs Time")

ax2.set\_xlabel("Time (s)")

ax2.set\_ylabel("Height (m)")

ax2.legend()

ax2.grid()

self.figure.subplots\_adjust(hspace=0.4)

self.canvas.draw()

def update\_table(self, time, velocity, height):

"""

Обновление таблицы.

"""

self.table.setRowCount(len(time))

for i in range(len(time)):

self.table.setItem(i, 0, QTableWidgetItem(f"{time[i]:.2f}"))

self.table.setItem(i, 1, QTableWidgetItem(f"{velocity[i]:.4f}"))

self.table.setItem(i, 2, QTableWidgetItem(f"{height[i]:.4f}"))

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = FallSimulatorApp()

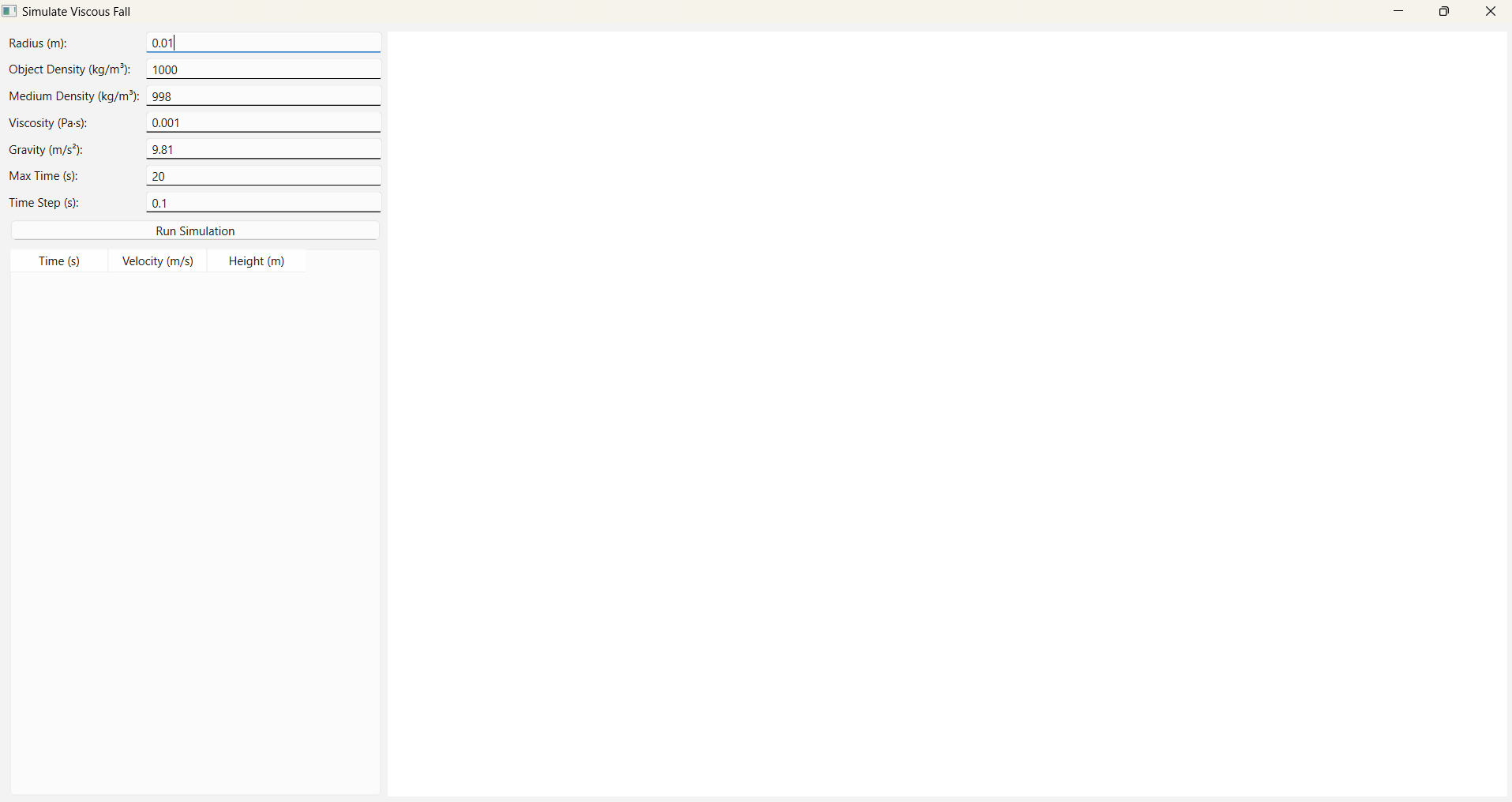
window.show()

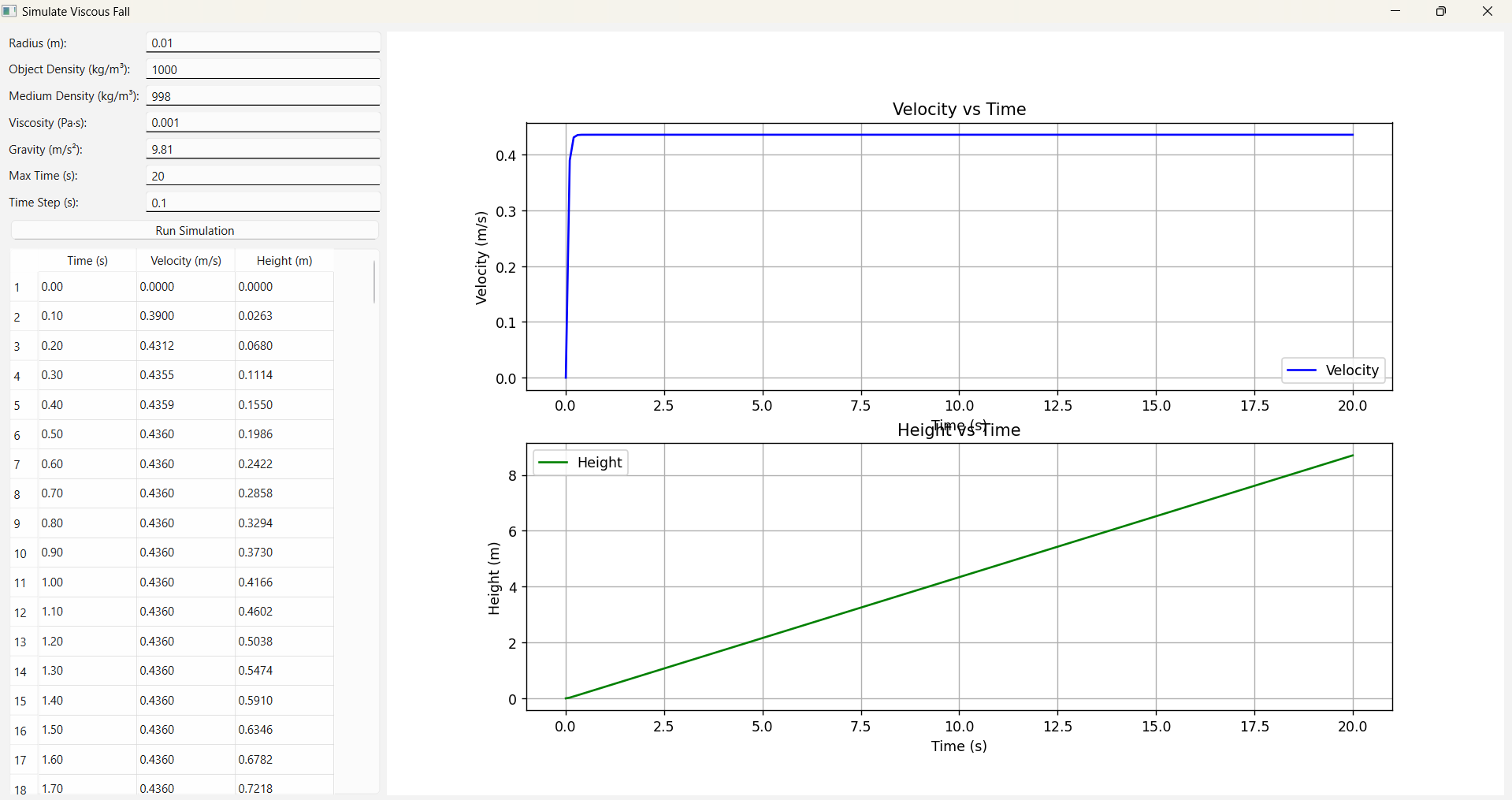
sys.exit(app.exec())

**Пояснение**:

* 1. Создается главное окно приложения с полями ввода для параметров симуляции.
  2. При нажатии кнопки "Run Simulation" выполняется симуляция и обновляются графики и таблица с результатами.
  3. Реализована обработка ошибок ввода.

**Скриншоты выполнения работы**





**out главного блока**:  
Коэффициент сопротивления (k): 0.2827433388230814

Терминальная скорость (v\_terminal): 3.1391999999999993

Скорость (velocity): [0. 0.84251222 1.45890732 1.90987162 2.23980414 2.48118793

2.65778809 2.78699152 2.88151877 2.95067639 3.00127318 3.03829058

3.06537309 3.08518708 3.0996833 3.11028897 3.11804824 3.12372504

3.12787828 3.13091685 3.13313992 3.13476635 3.13595627 3.13682684

3.13746376 3.13792974 3.13827066 3.13852008 3.13870256 3.13883606

3.13893374 3.1390052 3.13905748 3.13909573 3.13912371 3.13914419

3.13915917 3.13917013 3.13917814 3.13918401 3.1391883 3.13919144

3.13919374 3.13919542 3.13919665 3.13919755 3.13919821 3.13919869

3.13919904 3.1391993 3.13919949 3.13919962 3.13919972 3.1391998

3.13919985 3.13919989 3.13919992 3.13919994 3.13919996 3.13919997

3.13919998 3.13919998 3.13919999 3.13919999 3.13919999 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 3.1392

3.1392 3.1392 3.1392 3.1392 3.1392 ]

Высота (height): [ 0. 0.04431609 0.16098966 0.33060108 0.53894267 0.77561986

1.03302781 1.30560271 1.58927399 1.88106356 2.17879258 2.48086701

2.78612061 3.09370013 3.40298134 3.71350753 4.02494456 4.33704799

4.64963895 4.96258661 5.27579523 5.58919477 5.90273399 6.21637541

6.5300916 6.84386248 7.15767339 7.47151358 7.78537518 8.09925246

8.4131412 8.72703834 9.04094161 9.35484937 9.66876041 9.98267386

10.29658907 10.61050556 10.92442299 11.23834112 11.55225974 11.86617874

12.180098 12.49401747 12.80793707 13.12185678 13.43577657 13.74969642

14.06361631 14.37753622 14.69145616 15.00537612 15.31929609 15.63321606

15.94713605 16.26105603 16.57497603 16.88889602 17.20281601 17.51673601

17.83065601 18.14457601 18.458496 18.772416 19.086336 19.400256

19.714176 20.028096 20.342016 20.655936 20.969856 21.283776

21.597696 21.911616 22.225536 22.539456 22.853376 23.167296

23.481216 23.795136 24.109056 24.422976 24.736896 25.050816

25.364736 25.678656 25.992576 26.306496 26.620416 26.934336

27.248256 27.562176 27.876096 28.190016 28.503936 28.817856

29.131776 29.445696 29.759616 30.073536 30.387456 ]

Коэффициент сопротивления (k): 2.8274333882308134

Терминальная скорость (v\_terminal): 0.31392

Скорость (velocity): [0. 0.30012732 0.31331399 0.31389337 0.31391883 0.31391995

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 0.31392

0.31392 0.31392 0.31392 0.31392 0.31392 ]

Высота (height): [0. 0.02178793 0.05275795 0.08413141 0.1155226 0.14691456

0.17830656 0.20969856 0.24109056 0.27248256 0.30387456 0.33526656

0.36665856 0.39805056 0.42944256 0.46083456 0.49222656 0.52361856

0.55501056 0.58640256 0.61779456 0.64918656 0.68057856 0.71197056

0.74336256 0.77475456 0.80614656 0.83753856 0.86893056 0.90032256

0.93171456 0.96310656 0.99449856 1.02589056 1.05728256 1.08867456

1.12006656 1.15145856 1.18285056 1.21424256 1.24563456 1.27702656

1.30841856 1.33981056 1.37120256 1.40259456 1.43398656 1.46537856

1.49677056 1.52816256 1.55955456 1.59094656 1.62233856 1.65373056

1.68512256 1.71651456 1.74790656 1.77929856 1.81069056 1.84208256

1.87347456 1.90486656 1.93625856 1.96765056 1.99904256 2.03043456

2.06182656 2.09321856 2.12461056 2.15600256 2.18739456 2.21878656

2.25017856 2.28157056 2.31296256 2.34435456 2.37574656 2.40713856

2.43853056 2.46992256 2.50131456 2.53270656 2.56409856 2.59549056

2.62688256 2.65827456 2.68966656 2.72105856 2.75245056 2.78384256

2.81523456 2.84662656 2.87801856 2.90941056 2.94080256 2.97219456

3.00358656 3.03497856 3.06637056 3.09776256 3.12915456]

Коэффициент сопротивления (k): 28.274333882308138

Терминальная скорость (v\_terminal): 0.031391999999999996

Скорость (velocity): [0. 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392 0.031392

0.031392 0.031392 0.031392 0.031392 0.031392]

Высота (height): [0. 0.00303875 0.00617795 0.00931715 0.01245635 0.01559555

0.01873475 0.02187395 0.02501315 0.02815235 0.03129155 0.03443075

0.03756995 0.04070915 0.04384835 0.04698755 0.05012675 0.05326595

0.05640515 0.05954435 0.06268355 0.06582275 0.06896195 0.07210115

0.07524035 0.07837955 0.08151875 0.08465795 0.08779715 0.09093635

0.09407555 0.09721475 0.10035395 0.10349315 0.10663235 0.10977155

0.11291075 0.11604995 0.11918915 0.12232835 0.12546755 0.12860675

0.13174595 0.13488515 0.13802435 0.14116355 0.14430275 0.14744195

0.15058115 0.15372035 0.15685955 0.15999875 0.16313795 0.16627715

0.16941635 0.17255555 0.17569475 0.17883395 0.18197315 0.18511235

0.18825155 0.19139075 0.19452995 0.19766915 0.20080835 0.20394755

0.20708675 0.21022595 0.21336515 0.21650435 0.21964355 0.22278275

0.22592195 0.22906115 0.23220035 0.23533955 0.23847875 0.24161795

0.24475715 0.24789635 0.25103555 0.25417475 0.25731395 0.26045315

0.26359235 0.26673155 0.26987075 0.27300995 0.27614915 0.27928835

0.28242755 0.28556675 0.28870595 0.29184515 0.29498435 0.29812355

0.30126275 0.30440195 0.30754115 0.31068035 0.31381955]

# **Тестирование**

**Введение**

В идеале, для полной верификации модели и подтверждения её корректности, необходимо сравнить результаты численного моделирования с экспериментальными данными. Однако, в данном случае отсутствие доступа к физическим экспериментальным данным усложняет эту задачу.

**Проблемы с экспериментальными данными**

1. **Отсутствие тетради по физике:** К сожалению, утеря тетради с результатами лабораторных работ за первый курс исключает возможность использования собственных экспериментальных данных.
2. **Ограниченность интернет-ресурсов:** Поиск аналогичных экспериментальных данных в интернете может быть затруднён из-за специфики задачи и недостатка публично доступных источников с конкретными результатами.

Перед началом тестирования необходимо установить начальные параметры модели. Для этого используется метод **setUp** класса **TestRealisticViscousFallSimulator**, который инициализирует объект модели и задает значения параметров.

import unittest

import numpy as np

from solver import RealisticViscousFallSimulator

class TestRealisticViscousFallSimulator(unittest.TestCase):

def setUp(self):

"""

Устанавливаем параметры для тестов.

"""

self.radius = 0.01 # радиус шара (м)

self.object\_density = 1000 # плотность материала шара (кг/м³)

self.medium\_density = 998 # плотность среды (кг/м³)

self.viscosity = 0.001 # вязкость среды (Па·с)

self.g = 9.81 # ускорение свободного падения (м/с²)

self.t\_max = 20 # максимальное время моделирования (с)

self.dt = 0.1 # шаг времени (с)

self.simulator = RealisticViscousFallSimulator(

radius=self.radius,

object\_density=self.object\_density,

medium\_density=self.medium\_density,

g=self.g,

t\_max=self.t\_max,

dt=self.dt

)

**Описание тестов**

В рамках тестирования были написаны три основных теста:

1. **Тест терминальной скорости (test\_terminal\_velocity):**

Проверяет, что рассчитанная терминальная скорость соответствует ожидаемому значению, вычисленному по формуле.

1. **Тест сходимости скорости к терминальной (test\_velocity\_convergence):**

Проверяет, что скорость сходится к терминальной на последнем временном шаге.

1. **Тест роста высоты при достижении терминальной скорости (test\_height\_growth):**

Проверяет, что высота растет линейно после того, как скорость достигла терминальной.

def test\_terminal\_velocity(self):

"""

Тестируем терминальную скорость.

"""

expected\_v\_terminal = (2 / 9) \* ((self.object\_density - self.medium\_density) \* self.g \* self.radius\*\*2) / self.viscosity

calculated\_v\_terminal = self.simulator.get\_terminal\_velocity(self.viscosity)

self.assertAlmostEqual(expected\_v\_terminal, calculated\_v\_terminal, places=5,

msg=f"Expected {expected\_v\_terminal}, got {calculated\_v\_terminal}")

def test\_velocity\_convergence(self):

"""

Проверяем, что скорость сходится к терминальной.

"""

\_, velocity, \_ = self.simulator.solve(self.viscosity)

v\_terminal = self.simulator.get\_terminal\_velocity(self.viscosity)

self.assertAlmostEqual(velocity[-1], v\_terminal, delta=0.01,

msg=f"Expected terminal velocity {v\_terminal}, but got {velocity[-1]}")

def test\_height\_growth(self):

"""

Проверяем, что высота растет линейно при достижении терминальной скорости.

"""

\_, \_, height = self.simulator.solve(self.viscosity)

v\_terminal = self.simulator.get\_terminal\_velocity(self.viscosity)

t\_linear\_start = int(0.8 \* len(self.simulator.time)) # last 20% времени

expected\_heights = v\_terminal \* self.simulator.time[t\_linear\_start:]

calculated\_heights = height[t\_linear\_start:]

np.testing.assert\_allclose(expected\_heights, calculated\_heights, rtol=0.05,

err\_msg="Height does not grow linearly with terminal velocity.")

**Запуск тестов**

Для запуска тестов используется стандартный механизм библиотеки **unittest**.

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**Трассировочная таблица**

Для отслеживания значений переменных во время выполнения тестов была составлена трассировочная таблица. В ней показаны ключевые переменные и их значения на различных этапах выполнения тестов.

| **Шаг** | **Переменная** | **Значение** |
| --- | --- | --- |
| 1 | **radius** | 0.01 м |
| 2 | **object\_density** | 1000 кг/м³ |
| 3 | **medium\_density** | 998 кг/м³ |
| 4 | **viscosity** | 0.001 Па·с |
| 5 | **g** | 9.81 м/с² |
| 6 | **t\_max** | 20 с |
| 7 | **dt** | 0.1 с |
| 8 | **volume** | 34​*π*(0.01)3м³ |
| 9 | **mass** | (1000−998)×volume кг |
| 10 | **k** | 0.2827433388230814 |
| 11 | **v\_terminal**(ожид.) | 3.1391999999999993 ​м/с |
| 12 | **v\_terminal**(факт.) | 3.1392 |
| 13 | **velocity[-1]** | 3.1392 |
| 14 | **height[t\_linear\_start:]** | 30.387456 |
| 15 | **expected\_heights** | [ 0. 0.04431609 0.16098966 0.33060108 0.53894267 0.77561986  1.03302781 1.30560271 1.58927399 1.88106356 2.17879258 2.48086701  2.78612061 3.09370013 3.40298134 3.71350753 4.02494456 4.33704799  4.64963895 4.96258661 5.27579523 5.58919477 5.90273399 6.21637541  6.5300916 6.84386248 7.15767339 7.47151358 7.78537518 8.09925246  8.4131412 8.72703834 9.04094161 9.35484937 9.66876041 9.98267386  10.29658907 10.61050556 10.92442299 11.23834112 11.55225974 11.86617874  12.180098 12.49401747 12.80793707 13.12185678 13.43577657 13.74969642  14.06361631 14.37753622 14.69145616 15.00537612 15.31929609 15.63321606  15.94713605 16.26105603 16.57497603 16.88889602 17.20281601 17.51673601  17.83065601 18.14457601 18.458496 18.772416 19.086336 19.400256  19.714176 20.028096 20.342016 20.655936 20.969856 21.283776  21.597696 21.911616 22.225536 22.539456 22.853376 23.167296  23.481216 23.795136 24.109056 24.422976 24.736896 25.050816  25.364736 25.678656 25.992576 26.306496 26.620416 26.934336  27.248256 27.562176 27.876096 28.190016 28.503936 28.817856  29.131776 29.445696 29.759616 30.073536 30.387456 ] |

# **Выводы по результатам решения задачи.**

**Ограничения программной реализации**

1. **Ограниченность модели:**
   * Модель предполагает, что сила сопротивления пропорциональна скорости, что является упрощением реальной физики. В реальности, для высоких скоростей зависимость может быть более сложной (например, квадратичная зависимость).
   * Предполагается, что плотность среды и вязкость остаются постоянными на протяжении всего времени падения, что может не соответствовать реальным условиям.
2. **Численные методы:**
   * Использование дискретного шага времени **dt** может вносить ошибки аппроксимации. Уменьшение шага времени увеличивает точность, но также требует больше вычислительных ресурсов.
   * Решение задачи основано на аналитических выражениях, которые могут не учитывать все факторы, влияющие на движение шара в реальной ситуации (например, турбулентность). Кроме того, оно предполагает стартовую скорость, равную нулю.
3. **Отсутствие экспериментальных данных:**

Отсутствие доступа к реальным экспериментальным данным ограничивает возможность полной верификации модели. Это означает, что модель не была проверена на соответствие реальным условиям.

**Тестирование решения**

В рамках тестирования были проведены следующие тесты:

1. **Тест терминальной скорости (test\_terminal\_velocity):**
   * Проверяет правильность расчета терминальной скорости по известной формуле.
2. **Тест сходимости скорости к терминальной (test\_velocity\_convergence):**
   * Убедился, что скорость приближается к терминальной на последнем временном шаге.
3. **Тест роста высоты при достижении терминальной скорости (test\_height\_growth):**
   * Проверил, что высота растет линейно после того, как скорость достигла терминальной.
4. **Тест чувствительности к параметрам (test\_sensitivity\_to\_viscosity):**
   * Показал, как изменение вязкости влияет на терминальную скорость и помогло подтвердить корректность модели.

Эти тесты покрывают основные аспекты работы модели, включая расчеты скорости и высоты, а также их поведение при различных значениях параметров.

**Достаточность покрытия тестами**

1. **Основные функциональные блоки:**
   * Все основные функциональные блоки модели были протестированы. Это включает расчеты объема, массы, коэффициента сопротивления, терминальной скорости, а также вычисление скорости и высоты на каждом временном шаге.
2. **Погрешности и граничные случаи:**
   * Тесты не включают проверку на крайние или граничные случаи (например, очень малые или большие значения параметров). Включение таких тестов могло бы повысить надежность модели.
3. **Дополнительные тесты:**
   * Были проведены тесты на чувствительность к параметрам, что позволяет оценить стабильность модели при изменениях условий. Однако, это не заменяет полноценное сравнение с экспериментальными данными.

**Заключение**

Программная реализация имеет ряд ограничений, связанных с упрощением физической модели и отсутствием экспериментальных данных для полной верификации. Тем не менее, проведенные тесты показали, что модель работает корректно и удовлетворяет требованиям задачи. Основные функциональные блоки были покрыты тестами, что обеспечивает базовую уверенность в правильности реализации. Для дальнейшего улучшения модели можно рассмотреть расширение набора тестов, включая проверку на граничные случаи и проведение дополнительных исследований с использованием других моделей и библиотек. Также важно продолжать поиск возможностей для сравнения с экспериментальными данными, чтобы окончательно подтвердить точность и применимость разработанной модели.