# Отчет по выполнению моделирования задания "Лунолет"

## Введение

В рамках лабораторной работы по моделированию движения лунолета была поставлена задача разработать симуляцию безопасной посадки лунного корабля на поверхность Луны. Основная цель заключалась в определении высоты, на которой необходимо включить двигатель, чтобы вертикальная посадочная скорость не превышала 3 м/с. Для реализации задачи использовались язык программирования C#, технология WPF для создания пользовательского интерфейса и библиотека OxyPlot для построения графиков.

## Цель работы

Разработать программу, которая моделирует вертикальное движение лунолета с учетом включения двигателя для обеспечения безопасной посадки. Необходимо определить высоту, на которой должен быть включен двигатель, чтобы конечная вертикальная скорость при посадке не превышала 3 м/с.

## Постановка задачи

Инженер-электронщик лунной базы Иванов, тестируя автопилот малого лунного корабля, случайно включил двигатель, что привело к взлету корабля. После восстановления сознания Иванов обнаружил, что корабль находится на высоте $H_0$=2300м и движется со скоростью $V_{0y}$=20 м/с вниз. Необходимо рассчитать высоту, на которой необходимо включить двигатель, чтобы обеспечить безопасную посадку с вертикальной скоростью $V_{land} \leq 3$ м/с.

## Используемые константы и исходные данные

- **Ускорение свободного падения на Луне:** g=1.62 м/с²
- **Масса корабля без топлива:** M=2150 кг (включая пилота и скафандр)
- **Масса топлива:** $m_{fuel}$ =150 кг
- **Скорость истечения продуктов сгорания двигателя:** $V_p$=3660 м/с
- **Расход топлива двигателем:** m =15 кг/с
- **Начальная высота:** $H_0$=2300 м
- **Начальная скорость:** $V_{0y}$=20 м/с (вниз положительная)
- **Максимальная допустимая скорость при посадке:** $V_{max}$=3 м/с

## Методология

### Упрощения модели

1. **Двухмерное движение:** Рассматривается только вертикальное движение корабля.
2. **Плоская поверхность Луны:** Поверхность Луны принимается плоской, что упрощает расчеты.

3. **Непрерывное расходование топлива:** Предполагается, что двигатель работает непрерывно с постоянным расходом топлива.

### Основные уравнения

1. **Уравнение движения без включения двигателя (свободное падение):**

$$V(t+\Delta t)=V(t)+g$$

$$H(t+\Delta t)=H(t)-V(t)\cdot\Delta t-1/2g\cdot(\Delta t)^2$$

**Уравнение движения с включенным двигателем (управляемое снижение):**

$$a_{thrust}=-V_p\cdot m/m(t)$$

$$a_{total}=a_{thrust}-g$$

$$V(t+\Delta t)=V(t)+a_{total}\cdot\Delta t$$

$$H(t+\Delta t)=H(t)-V(t)\cdot\Delta t-1/2a_{total}\cdot(\Delta t)^2$$

$$m(t+\Delta t)=m(t)-m\cdot\Delta t$$

### Алгоритм моделирования

1. **Свободное падение:** Моделируется движение корабля до момента включения двигателя. На каждом временном шаге рассчитываются новая скорость и высота.
2. **Включение двигателя:** Перебором определяем момент, начиная с конца свободного падения, при котором включение двигателя обеспечивает достижение требуемой скорости посадки.
3. **Управляемое снижение:** После включения двигателя моделируется движение с учетом ускорения от двигателя и гравитации. Расчеты продолжаются до достижения поверхности Луны или достижения максимально допустимой скорости посадки.

### Выбор высоты включения двигателя

Для определения высоты, на которой необходимо включить двигатель, проводится итерация по высотам свободного падения и проверяется, достигает ли конечная скорость посадки требуемого значения.

# Реализация

### Используемые инструменты и технологии

- **Язык программирования:** C#
- **Платформа:** WPF (Windows Presentation Foundation)
- **Библиотека для построения графиков:** OxyPlot

### Описание основных компонентов

1. **Класс `SimulationResult`:** Хранит результаты симуляции, включая массивы времени, высоты, скорости, ускорения, а также конечную высоту включения двигателя и конечную скорость при посадке.
2. **Класс `LunarLanderSimulator`:** Отвечает за проведение симуляции. Включает методы для моделирования свободного падения и управляемого снижения.
3. **Класс `MainWindow`:** Отвечает за взаимодействие с пользователем, отображение графиков и результатов симуляции.

### Пользовательский интерфейс

Интерфейс приложения разработан с использованием технологии WPF и включает следующие элементы:

1. **Панель инструментов (Toolbar):** Содержит кнопки для запуска и сброса симуляции.
2. **Вкладки (TabControl):** Организованы три вкладки для отображения графиков зависимости высоты, скорости и ускорения от времени.
3. **Результаты симуляции:** Отображаются в отдельной области с выводом высоты включения двигателя и конечной скорости посадки.

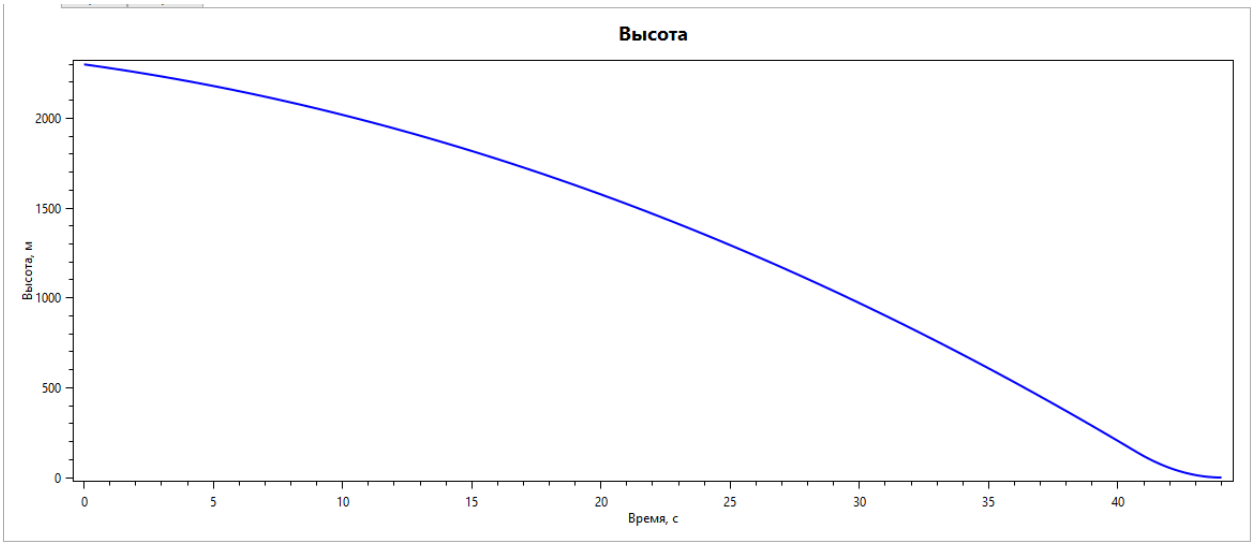
# Результаты

После выполнения симуляции были получены следующие результаты:

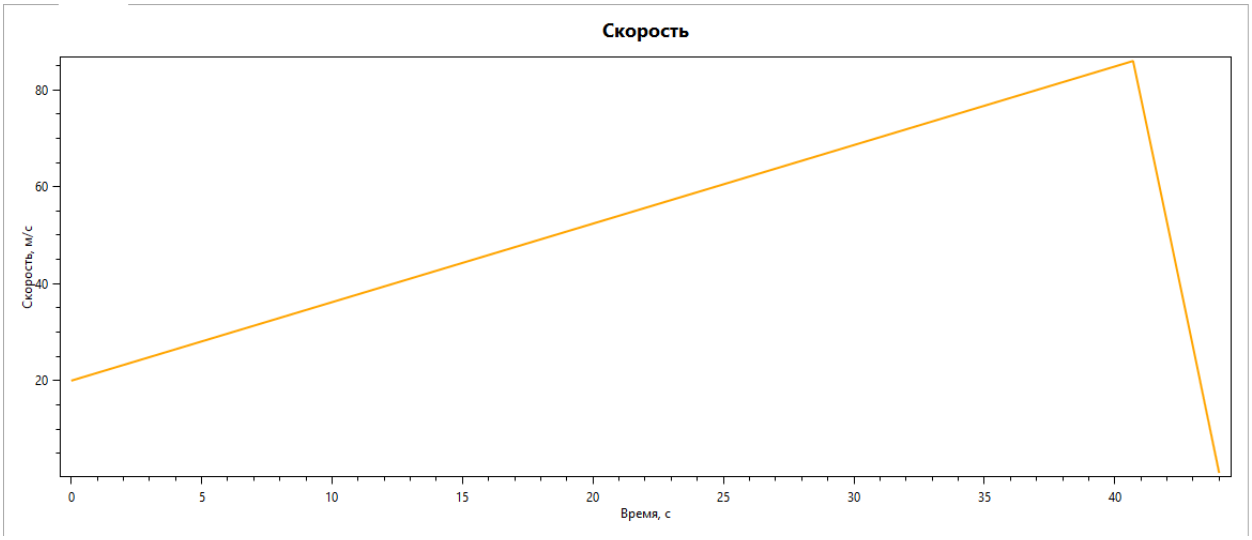
1. **Подходящая высота включения двигателя:** 144.24 м
2. **Вертикальная скорость при посадке:** 0.98 м/с

**Графики зависимости:**

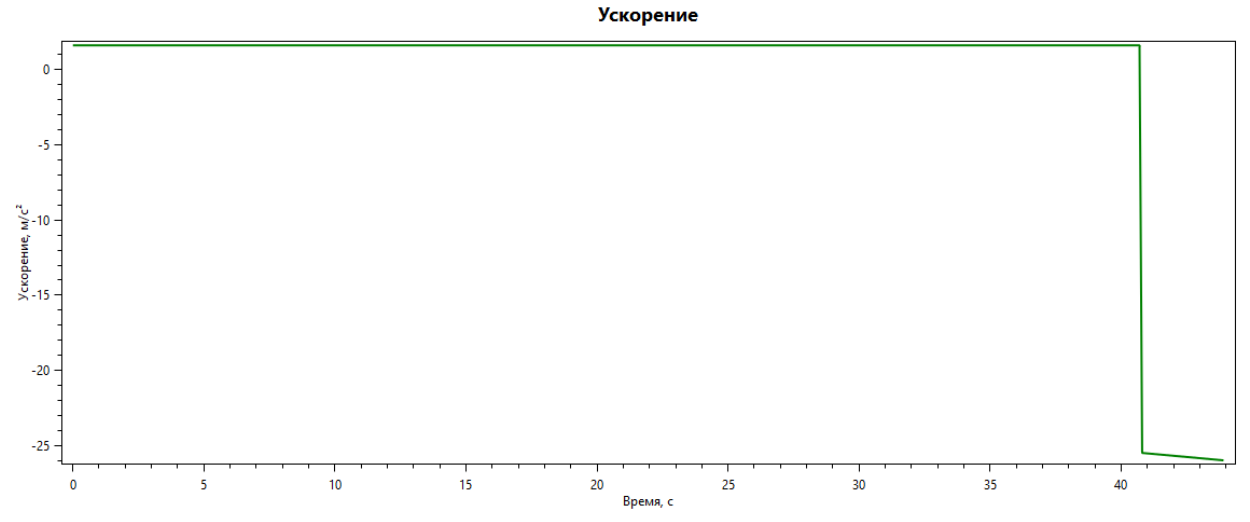
1. **Высота от времени:**



2. **Скорость от времени:**




3. **Ускорение от времени**




# Заключение

В ходе выполнения лабораторной работы была разработана и реализована модель симуляции посадки лунолета на поверхность Луны с использованием языка C#, технологии WPF и библиотеки OxyPlot. Модель позволяет определить оптимальную высоту включения двигателя для обеспечения безопасной посадки с вертикальной скоростью, не превышающей 3 м/с. Полученные результаты соответствуют поставленным требованиям, что подтверждает корректность реализации модели.

# Приложения

## Исходный код

**`SimulationResult.cs`**

```
public class SimulationResult
{
    public double[] Time { get; set; }
    public double[] Height { get; set; }
    public double[] Velocity { get; set; }
    public double[] Acceleration { get; set; }
    public double LandingHeight { get; set; }
    public double LandingVelocity { get; set; }
    public bool IsSuccessful { get; set; }
    public string Message { get; set; }
}
```

**LunarLanderSimulator.cs**

```
public class LunarLanderSimulator
{
    // Константы
    private const double Gravity = 1.62; // м/с²
    private const double MassEmpty = 2150; // кг
    private const double InitialFuelMass = 150; // кг
    private const double FuelConsumptionRate = 15; // кг/с
    private const double ExhaustVelocity = 3660; // м/с
    private const double InitialHeight = 2300; // м
    private const double InitialVelocity = 20; // м/с (вниз положительная)
    private const double MaxLandingSpeed = 3; // м/с
    private const double DeltaTime = 0.1; // с

    public SimulationResult RunSimulation()
    {
        // Моделирование свободного падения
        var free = FreeFall(InitialHeight, InitialVelocity, Gravity,
DeltaTime);
        double[] t_free = free.t;
        double[] H_free = free.H;
        double[] V_free = free.V;

        bool engineStarted = false;
        double[] t_total = null;
        double[] H_total = null;
        double[] V_total = null;
        double[] a_total = null;
        double H_turn_on = 0;
        double V_turn_on = 0;

        // Поиск высоты включения двигателя
        for (int idx = t_free.Length - 1; idx >= 0; idx--)
        {
            if (H_free[idx] <= 0)
                continue;

            H_turn_on = H_free[idx];
```

```
            V_turn_on = V_free[idx];
            double initialMass = MassEmpty + InitialFuelMass;

            var powered = PoweredDescent(H_turn_on, V_turn_on, initialMass,
Gravity, ExhaustVelocity, FuelConsumptionRate, MaxLandingSpeed, DeltaTime);
            double[] t_powered = powered.Time;
            double[] H_powered = powered.Height;
            double[] V_powered = powered.Velocity;
            double[] a_powered = powered.Acceleration;

            if (V_powered.Last() <= MaxLandingSpeed)
            {
                engineStarted = true;
                t_total = t_free.Take(idx + 1)
                               .Concat(t_powered.Skip(1).Select(tp =>
t_free[idx] + tp))
                               .ToArray();
                H_total = H_free.Take(idx + 1)
                                .Concat(H_powered.Skip(1))
                                .ToArray();
                V_total = V_free.Take(idx + 1)
                                .Concat(V_powered.Skip(1))
                                .ToArray();
                double[] a_free = Enumerable.Repeat(Gravity, idx +
1).ToArray();
                a_total = a_free.Concat(a_powered).ToArray();
                break;
            }
        }

        var result = new SimulationResult();

        if (!engineStarted)
        {
            result.IsSuccessful = false;
            result.Message = "Не удалось найти подходящую высоту включения
двигателя для безопасной посадки.";
        }
        else
        {
            result.IsSuccessful = true;
            result.LandingHeight = H_turn_on;
            result.LandingVelocity = V_total.Last();
            result.Time = t_total;
            result.Height = H_total;
            result.Velocity = V_total;
            result.Acceleration = a_total;
        }

        return result;
    }
```

```
    private (double[] t, double[] H, double[] V) FreeFall(double H_initial,
double V_initial, double gravity, double delta_t)
    {
        var t_list = new List<double> { 0 };
        var H_list = new List<double> { H_initial };
        var V_list = new List<double> { V_initial };

        while (H_list.Last() > 0)
        {
            double V_new = V_list.Last() + gravity * delta_t;
            double H_new = H_list.Last() - V_list.Last() * delta_t - 0.5 *
gravity * Math.Pow(delta_t, 2);
            double t_new = t_list.Last() + delta_t;

            t_list.Add(t_new);
            V_list.Add(V_new);
            H_list.Add(H_new > 0 ? H_new : 0);

            if (H_new <= 0)
                break;
        }

        return (t_list.ToArray(), H_list.ToArray(), V_list.ToArray());
    }

    private (double[] Time, double[] Height, double[] Velocity, double[]
Acceleration) PoweredDescent(
        double H_start, double V_start, double m0, double gravity, double
exhaust_velocity, double fuel_rate,
        double max_speed, double delta_t)
    {
        var t_list = new List<double> { 0 };
        var H_list = new List<double> { H_start };
        var V_list = new List<double> { V_start };
        var a_list = new List<double>();

        double currentMass = m0;

        while (H_list.Last() > 0 && currentMass > MassEmpty)
        {
            double a_thrust = -exhaust_velocity * fuel_rate / currentMass;
            double a_total = a_thrust - gravity;
            a_list.Add(a_total);

            double V_new = V_list.Last() + a_total * delta_t;
            double H_new = H_list.Last() - V_list.Last() * delta_t - 0.5 *
a_total * Math.Pow(delta_t, 2);
            double t_new = t_list.Last() + delta_t;

            t_list.Add(t_new);
```

```
            V_list.Add(V_new);
            H_list.Add(H_new > 0 ? H_new : 0);
            currentMass -= fuel_rate * delta_t;

            if (H_new <= 0 || V_new <= max_speed)
                break;
        }

        return (t_list.ToArray(), H_list.ToArray(), V_list.ToArray(),
a_list.ToArray());
    }
}
```

**MainWindow.xaml.cs**

```
public partial class MainWindow : Window, INotifyPropertyChanged
{
    // Свойства для привязки графиков и текста результата
    private PlotModel heightPlotModel;
    public PlotModel HeightPlotModel
    {
        get => heightPlotModel;
        set
        {
            heightPlotModel = value;
            OnPropertyChanged(nameof(HeightPlotModel));
        }
    }

    private PlotModel speedPlotModel;
    public PlotModel SpeedPlotModel
    {
        get => speedPlotModel;
        set
        {
            speedPlotModel = value;
            OnPropertyChanged(nameof(SpeedPlotModel));
        }
    }

    private PlotModel accelerationPlotModel;
    public PlotModel AccelerationPlotModel
    {
        get => accelerationPlotModel;
        set
        {
            accelerationPlotModel = value;
            OnPropertyChanged(nameof(AccelerationPlotModel));
        }
    }
```

```
    private string resultText;
    public string ResultText
    {
        get => resultText;
        set
        {
            resultText = value;
            OnPropertyChanged(nameof(ResultText));
        }
    }

    public MainWindow()
    {
        InitializeComponent();
        DataContext = this;
        PerformSimulation();
    }

    // Реализация интерфейса INotifyPropertyChanged
    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged(string propertyName) =>
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));

    // Метод для выполнения симуляции
    private void PerformSimulation()
    {
        var simulator = new LunarLanderSimulator();
        SimulationResult result = simulator.RunSimulation();

        if (!result.IsSuccessful)
        {
            ResultText = result.Message;
        }
        else
        {
            // Создание моделей графиков
            HeightPlotModel = CreatePlotModel("Высота", "Время, с", "Высота,
м", result.Time, result.Height, OxyColors.Blue);
            SpeedPlotModel = CreatePlotModel("Скорость", "Время, с",
"Скорость, м/с", result.Time, result.Velocity, OxyColors.Orange);
            AccelerationPlotModel = CreatePlotModel("Ускорение", "Время, с",
"Ускорение, м/с²", result.Time.Take(result.Time.Length - 1).ToArray(),
result.Acceleration, OxyColors.Green);

            // Вывод результатов
            ResultText = $"Подходящая высота для включения двигателя:
{result.LandingHeight:F2} м\n" +
                         $"Вертикальная скорость при посадке:
{result.LandingVelocity:F2} м/с";
        }
```

```
    }

    // Метод для создания модели графика
    private PlotModel CreatePlotModel(string title, string xAxisTitle, string
yAxisTitle, double[] x, double[] y, OxyColor color)
    {
        var model = new PlotModel { Title = title };
        model.Axes.Add(new LinearAxis { Position = AxisPosition.Bottom, Title
= xAxisTitle });
        model.Axes.Add(new LinearAxis { Position = AxisPosition.Left, Title =
yAxisTitle });

        var series = new LineSeries
        {
            Title = title,
            Color = color,
            MarkerType = MarkerType.None
        };

        for (int i = 0; i < x.Length; i++)
        {
            series.Points.Add(new DataPoint(x[i], y[i]));
        }

        model.Series.Add(series);
        return model;
    }

    private void RunSimulation_Click(object sender, RoutedEventArgs e)
    {
        PerformSimulation();
    }

    private void ResetSimulation_Click(object sender, RoutedEventArgs e)
    {
        // Логика сброса симуляции
        HeightPlotModel = null;
        SpeedPlotModel = null;
        AccelerationPlotModel = null;
        ResultText = "Симуляция сброшена. Нажмите 'Запустить симуляцию' для
начала.";
    }
}
```