

# ОЛИМПИАДА ШКОЛЬНИКОВ «ШАГ В БУДУЩЕЕ»

## НАУЧНО-ОБРАЗОВАТЕЛЬНОЕ СОРЕВНОВАНИЕ «ШАГ В БУДУЩЕЕ, МОСКВА»

1474

*регистрационный номер*

ИУ-7

*название факультета*

Программное обеспечение ЭВМ и информационные технологии

*название кафедры*

Программа для ракетного комплекса

*название работы*

**Автор:**

Краснов Леонид

Антонович

*фамилия, имя, отчество*

Школа 1502 «Энергия», 11 М

*наименование учебного заведения, класс*

Зуев Максим Максимович

*фамилия, имя, отчество*

Школа 1502 «Энергия»

*место работы*

Учитель

*звание, должность*

*подпись научного руководителя*

**Научный руководитель:**

## АННОТАЦИЯ

Программа для ракетного комплекса используется для моделирования полета неуправляемой ракеты типа “Земля-земля”. В приложении учтен разгон по направляющей, изменяемая масса и угол относительно горизонта на активном участке траектории. Неучтенно сопротивление воздуха, шарообразность земли и изменение гравитации от высоты.

Программа является оконным приложением, которое принимает в поля для ввода текста параметры ракеты и условия запуска, выводит траекторию полета ракеты и угол запуска. Выводит конечные результаты, такие как высота подъема, дальность полета.

Программа предназначена для ракет, не выходящих на орбиту. У ракеты типа земля-земля выделяются 3 фазы полета:

1. Разгон по направляющей – ракета начинает свое движение по специальной рельсе, направленной вверх под некоторым углом к горизонту. Такая рельса используется для разгона ракеты. Без нее траектория движения ракеты становится менее предсказуемым;
2. Движение ракеты на активном участке траектории – ракета летит на двигателе, направляя вектор тяги по вектору скорости до тех пор, пока не кончится топливо;
3. Движение ракеты на пассивном участке траектории – ракета летит по инерции, пока не достигнет земли. На этом участке ракета рассматривается как тело, брошенное под углом к горизонту.

## Содержание

ВВЕДЕНИЕ .....	4
1 Основная часть .....	5
1.1 История создания.....	5
1.2 Постановка задачи .....	9
1.3 Анализ предметной области .....	10
1.4 Используемые методы.....	11
1.5 Алгоритмы и средства реализации .....	12
1.6 Предложения по практическому применению .....	19
1.7 Перспективы дальнейшей разработки .....	20
2 Программы - аналоги.....	21
2.1 LANE.....	21
2.2 Open rocket.....	22
3 Заключение .....	23
4 Список использованных источников .....	26
Приложение А. Листинг программ.....	27
Код файла phase.cpp .....	27
Код файла phase.h .....	29
Код файла numeric.cpp .....	30
Код файла numeric.h .....	32
Код файла model.cpp .....	33
Код файла model.h .....	35
Код файла inverse.cpp.....	36
Код файла inverse.h.....	36
Код файла counting.cpp .....	37
Код файла counting.h .....	43
Код файла MyForm.cpp .....	45
Код файла MyForm.h.....	45

## ВВЕДЕНИЕ

При свободном выборе темы для проекта у автора проявился интерес к ракетостроению. При углублении в эту тему возникла проблема в расчете соотношений массы ракеты, видах топлива, доступных для использования, способах запуска ракеты типа “Земля-земля” и нахождении траектории ракеты. Проект был предназначен для работы в команде, поэтому, был разделен на две части: Программную и техническую. Программную часть взял на себя автор, а техническую – напарник по проекту Федор Чернопятко. Так, задачей автора стало написание программы для расчёта траектории неуправляемой ракеты “Земля-земля”.

Приложение должно было принимать на вход простые данные, которые можно получить в “домашних условиях”, иметь возможность вывода нескольких траекторий для визуального сравнения, иметь вычисляемую и небольшую погрешность вычислений, так как она является накапливаемой.

В программе моделируются три участка полета ракеты – разгон по направляющей, активный участок траектории и пассивный участок траектории. Для каждого участка применяются свои методы. Так для разгона по направляющей был написан метод, для которого была выведена формула движения на этом участке. Для активного участка траектории было выведено дифференциальное уравнение, для решения которого был применен метод Рунге-Кутты 4-го порядка. Для пассивного участка траектории применяется закон для тела, брошенного под углом к горизонту.

Благодаря разбиению траектории на три участка и реализации методов в своих классах удалось сделать код программы более читаемым.

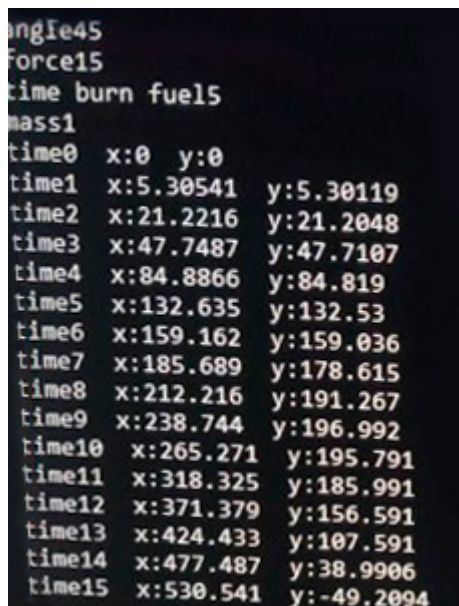
По ходу данной работы автор сталкивался и решал следующие проблемы: Разворот ракеты на активном участке, движение ракеты по направляющей, решение дифференциального уравнения, создание оконного приложения.

Код программы приведен в приложении А.

## 1 Основная часть

### 1.1 История создания

Проект имеет длинную историю создания. Сначала это было консольным приложением C++ (рисунок 1), которое выводило траекторию ракеты в виде координат точек. Масса ракеты была неизменной на всей траектории, не было направляющей, ракета не меняла угол относительно горизонта, из-за чего летала боком.



```
angle45
force15
time burn fuel5
mass1
time0 x:0 y:0
time1 x:5.30541 y:5.30119
time2 x:21.2216 y:21.2048
time3 x:47.7487 y:47.7107
time4 x:84.8866 y:84.819
time5 x:132.635 y:132.53
time6 x:159.162 y:159.036
time7 x:185.689 y:178.615
time8 x:212.216 y:191.267
time9 x:238.744 y:196.992
time10 x:265.271 y:195.791
time11 x:318.325 y:185.991
time12 x:371.379 y:156.591
time13 x:424.433 y:107.591
time14 x:477.487 y:38.9906
time15 x:530.541 y:-49.2094
```

Рисунок 1 – Консольное приложение

Далее возникла идея визуализировать эту траекторию, однако у автора не было времени осваивать win-api. В результате поиска конструктора для создания оконного приложения, такой конструктор обнаружился в шаблоне проекта C#, после чего проект преобразился в оконное приложение с удобным вводом и визуализацией траектории. Полем для графика являлся “холст”. Траектория, оси и масштабная сетка рисовались программой с помощью кисти (рисунок 2).

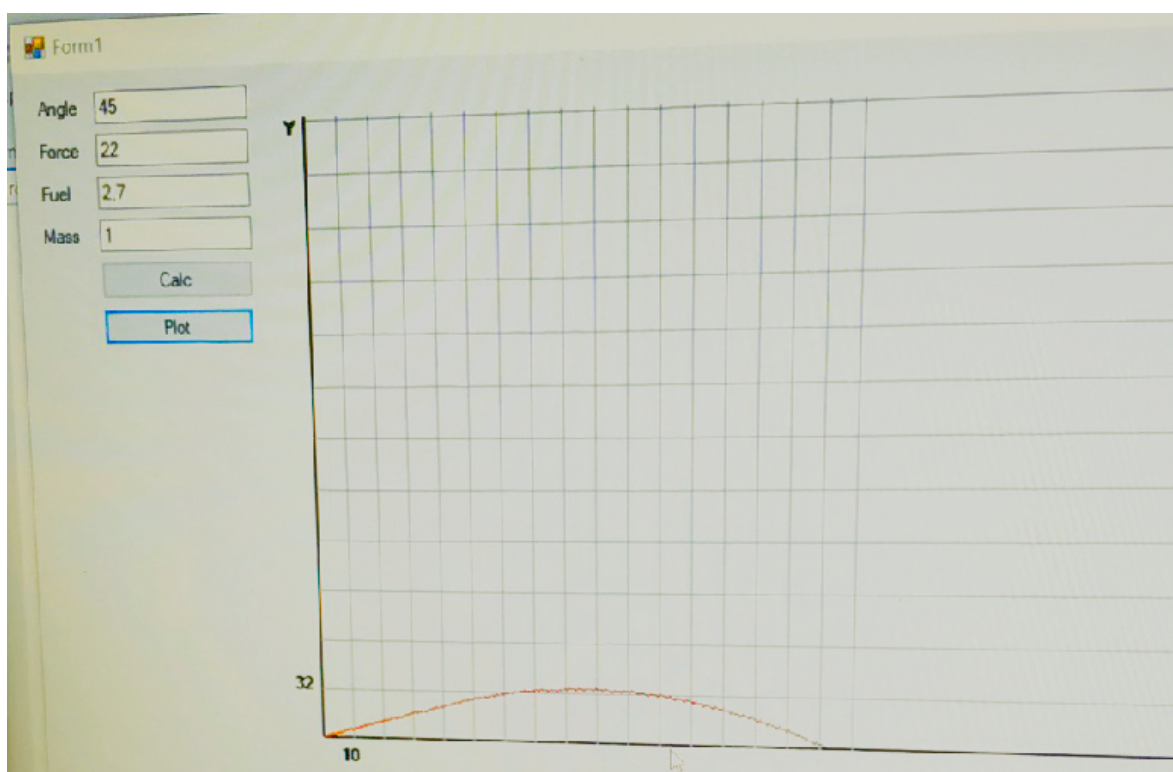


Рисунок 2 – Оконное приложение C#

Так как ракета не поворачивалась во время полета, то есть летала боком, было принято решение поворачивать ракету во время полета по вектору скорости. Для разворота во время полета было составлено дифференциальное уравнение, которое решалось методом прямоугольников. Тогда же после изучения видеоматериалов и литературы по любительскому ракетостроению в программу была добавлена направляющая для разгона (рисунок 3).

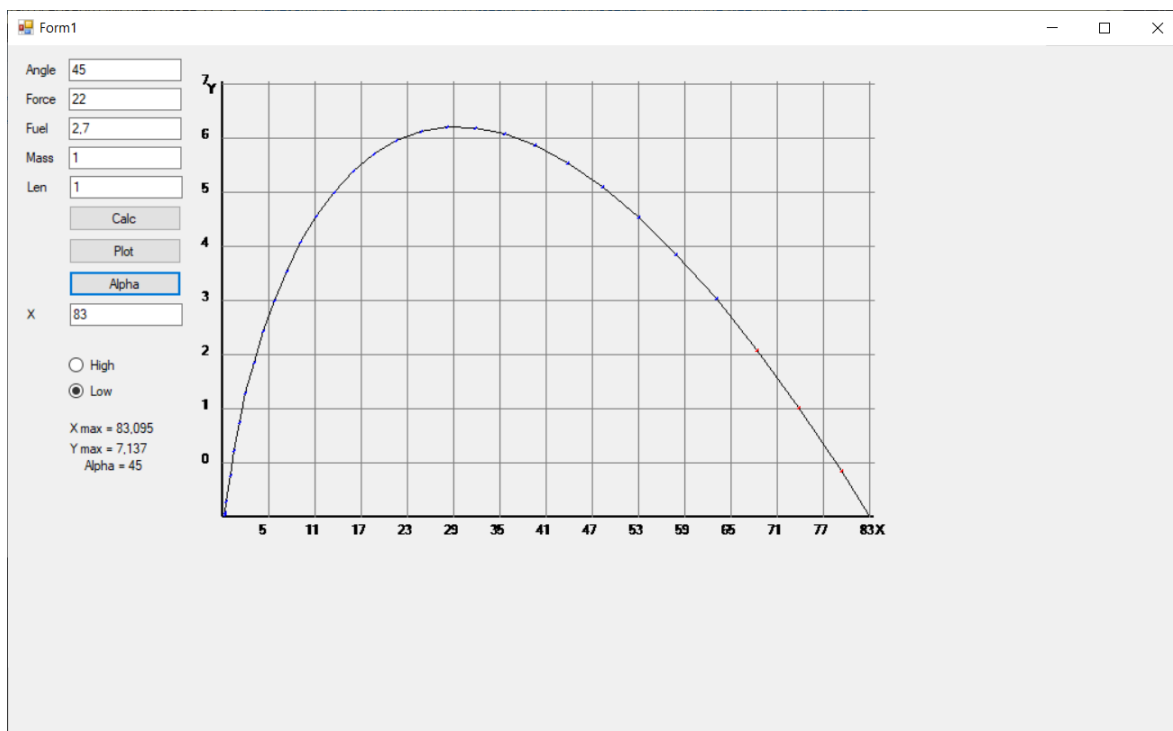


Рисунок 3 – Визуализация новой траектории ракеты

Язык C# был новым для автора, поэтому процесс разработки был долгим, так как многое приходилось искать в интернете. После тщательного поиска, было обнаружено, что у языка C++ то же есть конструктор оконных приложений, однако его надо было подключать отдельно (он не входил в шаблон). Так проект был перенесен на C++.

Во время изучения теории по созданию оконных приложений был обнаружен объект Chart, который принимает массив точек, а выводит график, состоящий из них, сам рисует оси и масштабную сетку.

Так же в программу была добавлена возможность посчитать угол, при котором ракета преодолет заданное пользователем расстояние.

После анализа траекторий, полученных в результате работы программы встала задача вставить в программу изменяемую массу ракеты. было принято решение – каждый раз, когда программа считает новую точку, она вычитала из массы топлива, оставшуюся на предыдущем шаге вычитала массу израсходованного топлива, которое сгорело за промежуток времени, однако такой метод имеет погрешность, и через некоторое время было составлено уравнение движения по направляющей, в которой учитывается расход топлива.

После тестов программы, оказалось, что вычисления имеют огромную накапливаемую погрешность, которая получалась из-за численного решения дифференциального уравнения методом прямоугольников. Это было обнаружено, когда во время программного эксперимента был изменен шаг вычисления новой точки.

Следующим шагом стало добавление в проект его символа (рисунок 4).

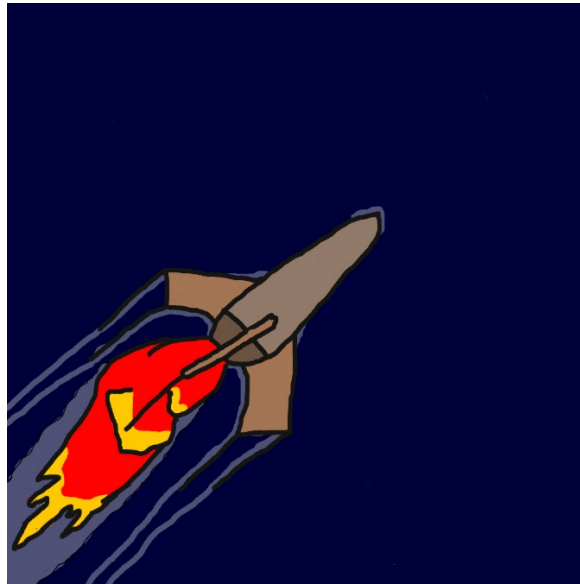


Рисунок 4 - Иконка

Для анализа разных траекторий была добавлена возможность рисовать несколько траекторий на одном поле (рисунок 5).

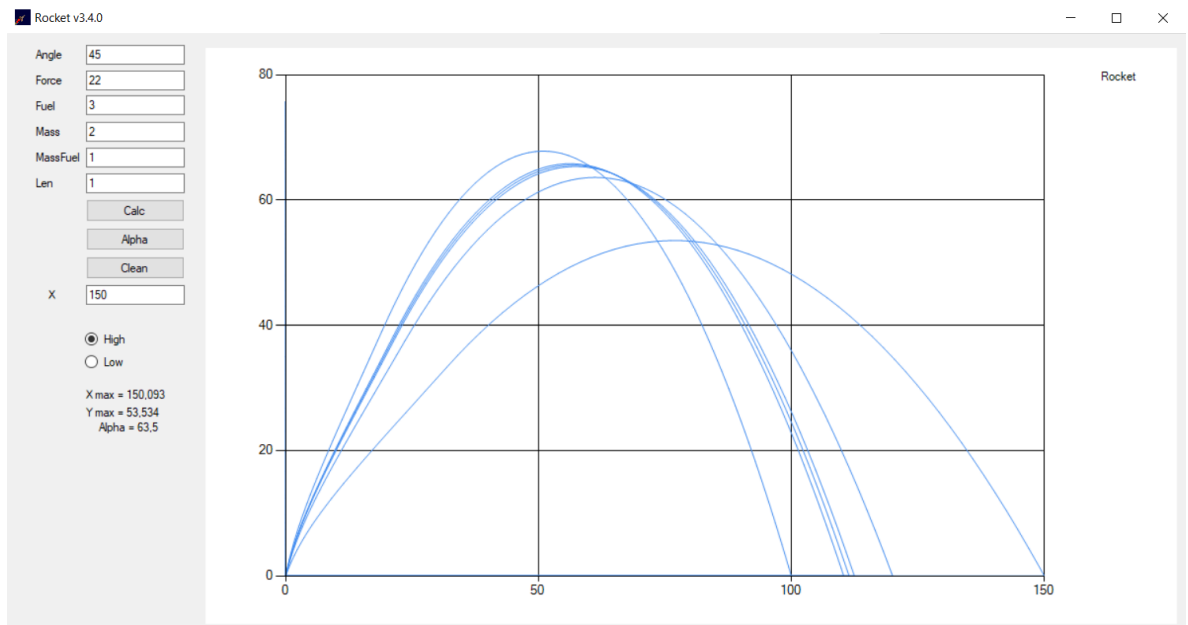


Рисунок 5- Визуализация нескольких траекторий



Было необходимо найти более точные методы для решения уравнения. Так был найден метод Рунге-Кутты 4-го порядка. Не удалось разобраться, как он работает, однако получилось вставить его в программу. С новым методом удалось посчитать погрешность и вывести ее график (рисунок 6).

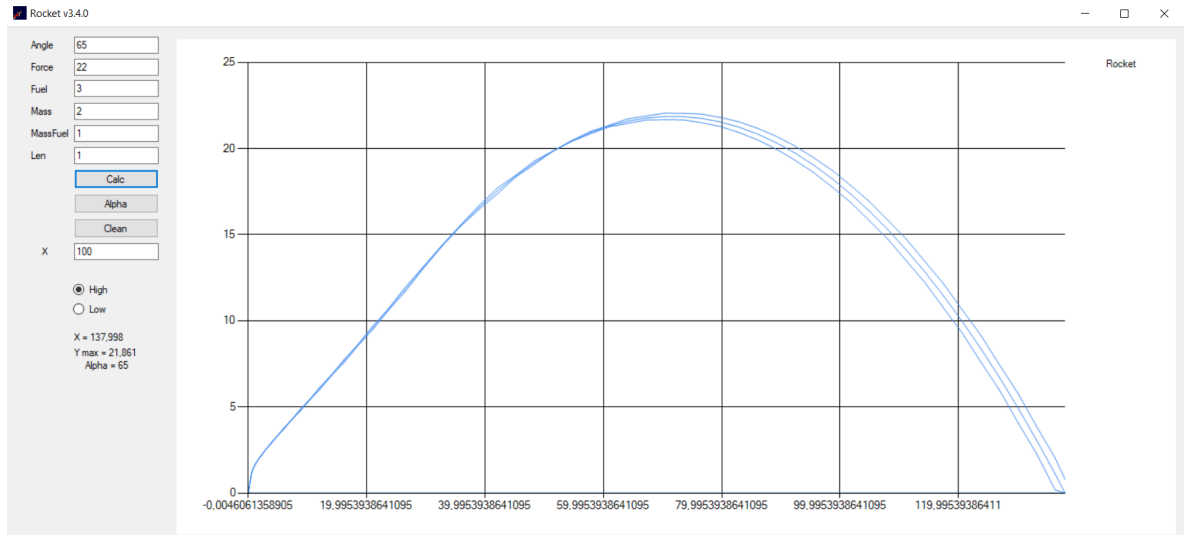


Рисунок 6 – Туннель погрешности

## 1.2 Постановка задачи

Было необходимо создать приложение для расчета и визуализации траектории ракеты, расчета угла запуска и нахождения расстояния до точки приземления и иметь незначительную погрешность, которое должно работать в двух режимах:

- рассчитывать и выводить на экран траекторию полета ракеты при заданном угле.
- Рассчитывать и выводить на экран траекторию полета ракеты при заданном расстоянии.

Пользователь вводит:

- Силу тяги двигателя.
- Вес заправленной ракеты.
- Массу топлива.
- Длину направляющей.

— Время горения топлива.

Далее, если искомым является угол, то пользователь вводит дальность полета.

В том случае, если искомым является дальность полета, пользователь вводит угол взлета.

В программе должна быть возможность вывода нескольких траекторий в поле для графика.

### 1.3 Анализ предметной области

Очевидно, что полет ракеты состоит из трех последовательных фаз:

Для расчета движения по направляющей была составлена формула (1), где путь является первообразной от скорости.

$$s = \int v \times dt = \frac{F}{\mu^2(\mu \times t - m_0)} \times \ln\left(\frac{m_0}{m_0 + \mu \times t}\right) + F \times \frac{t}{\mu} - g \times \sin(\alpha) \times \frac{t^2}{2}, \quad (1)$$

где:  $s$  – путь [м].

$v$  – скорость [м/с<sup>2</sup>].

$t$  – время [с].

$F$  – тяга двигателя [Н].

$\mu$  – расход топлива [кг/с].

$m_0$  – масса заправленной ракеты [кг].

$g$  – ускорение свободного падения [м/с<sup>2</sup>].

Для расчета траектории ракеты на активном участке была выведена система дифференциальных уравнений (2) и (3).

$$V^o x = \frac{V_x F}{M \sqrt{V^2 x + V^2 y}} \quad (2)$$

$$V^o y = \frac{V_y F}{M \sqrt{V^2 x + V^2 y}} - g \quad (3)$$

где  $V^o x$  – производная по времени от скорости по оси  $x$ , м/с<sup>2</sup>.

$M$  – масса ракеты на данный момент, кг.

$V^o y$  – производная по времени от скорости по оси  $y$ , м/с<sup>2</sup>.

$F$  – Тяга двигателя, Н.

$g$  – ускорение свободного падения,  $\text{м/с}^2$ .

Для расчета траектории ракеты на пассивном участке была применена формула для тела, брошенного под углом к горизонту.

$$x = v * \cos(\alpha) * t$$

$$y = v * \sin(\alpha) * t - (g * t^2)/2$$

где  $\alpha$  – угол, радианы.

$t$  – время, с.

$g$  – ускорение свободного падения,  $\text{м/с}^2$ .

$v$  – скорость,  $\text{м/с}$ .

$x$  – координата по оси  $X$ .

$y$  – координата по оси  $Y$ .

#### 1.4 Используемые методы

Для разработки приложения использовалась:

- Среда программирования Microsoft visual studio 2019 была выбрана так как у автора был опыт разработки в этой среде и есть конструктор windows forms.
- Язык программирования C++ был выбран, так как у автора есть опыт программирования на этом языке, у языка есть поддержка и большое русскоязычное сообщество, так же большое количество библиотек, которые используются в проекте написаны на C++.
- Стандартная библиотека шаблонов STL(C++) была включена в проект для использования класса vector и string. Они были необходимы для удобной работы с элементами и комфортной отладки программы.
- Объектно-ориентированное программирование было использовано для навигации по проекту.
- Метод Рунге-Кутты 4-го порядка для расчета траектории ракеты на активном участке.

— Правило Рунге для расчета погрешности от шага.

### 1.5 Алгоритмы и средства реализации

Функция `Cos` и `Sin` из библиотеки `cmath` в качестве аргумента принимают угол в радианах, поэтому приложение конвертирует угол из градусов в радианы.

В программе не учитывается сила сопротивления воздуха, а ракета представлена как материальная точка с вектором тяги и массой. Для того чтобы траектория ракеты была схожа с реальной, было принято решение разворачивать ракету по направлению вектора скорости.

Временной промежуток между соседними точками составляет 0.1 секунду. Точки записываются в массив типа `vector`

Пассивная фаза полета длится пока ракета не достигнет земли. Так как новая точка ставится каждую 0.1 секунду, то по оси `Y` ракета уходит в минус, для того, чтобы найти последнюю точку программа берет предпоследнюю и рассчитывает точку приземления исходя из высоты, скорости и ускорения ракеты.

Метод золотого сечения – используется в программе для нахождения максимальной дальности полета ракеты (рисунок 7).

На первой итерации заданный отрезок делится двумя симметричными относительно центра точками и рассчитываются значения для каждой из точек, после чего конец отрезка, ближе к которому оказалась та из вновь поставленных точек, значение которой больше переносится в эту точку. На следующей итерации, благодаря свойству метода золотого сечения надо искать всего одну новую точку. Эти действия повторяются, пока расстояние между двумя точками не станет меньше или равно поставленной точности. Далее, для увеличения точности, координаты двух точек складываются и делятся на 2. Полученная координата и будет точкой максимума.

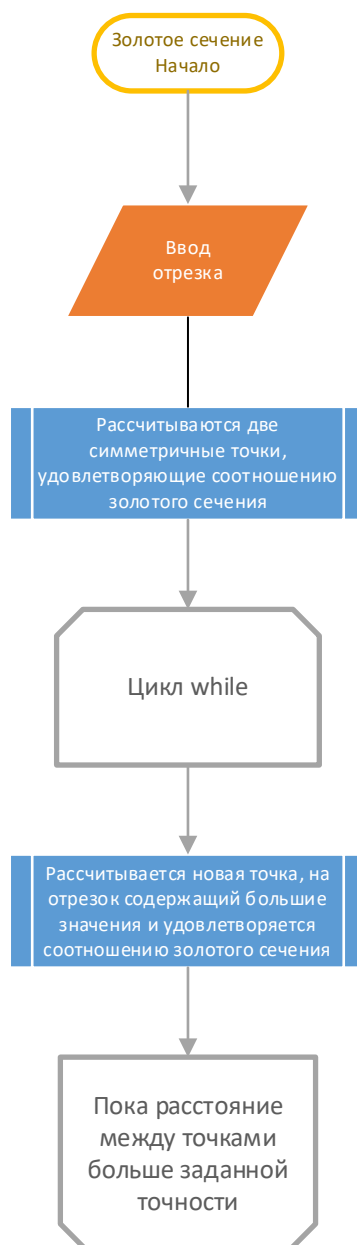


Рисунок 7 – Метод золотого сечения

Метод деления пополам – используется в программе для нахождения угла, при котором ракета преодолеет заданное пользователем расстояние. Программа вычисляет середину заданного отрезка, далее рассчитав интервал значений на двух отрезках, программа определяет на какой из двух частей лежит решение. Отрезок, в интервал которого не входит решение отбрасывается, а оставшийся делится пополам и происходят те же самые действия, пока длина отрезка не будет меньше или равно поставленной точности. Далее для увеличения точности координаты начала отрезка складываются с координатами

конца и делятся на 2. Полученная координата и есть угол запуска ракеты, при котором ракета улетит на заданное расстояние.

Для расчета погрешности от шага в методе Рунге-Кутты было применено правило Рунге. Метод работает так: траектория ракеты считается на двух масштабных сетках, один раз с шагом  $h$ , другой раз с шагом  $h/2$ . Далее из координаты, полученной на первой сетке вычитается координата полученная на второй, результат берется по модулю и делится на 15 (для метода Рунге-Кутта 4-го порядка). Таким образом получается график погрешности по оси  $Y$  по промежутку времени. Для наглядной визуализации данной погрешности было принято решение из координат траектории ракеты вычесть координаты траектории погрешности для получения графика нижней границы погрешности, а для нахождения верхней – складывать координаты траектории ракеты с координатами траектории погрешности. В результате проведенных действий получился “туннель” погрешности. Линия между двумя границами туннеля – траектория ракеты (рисунок 8).

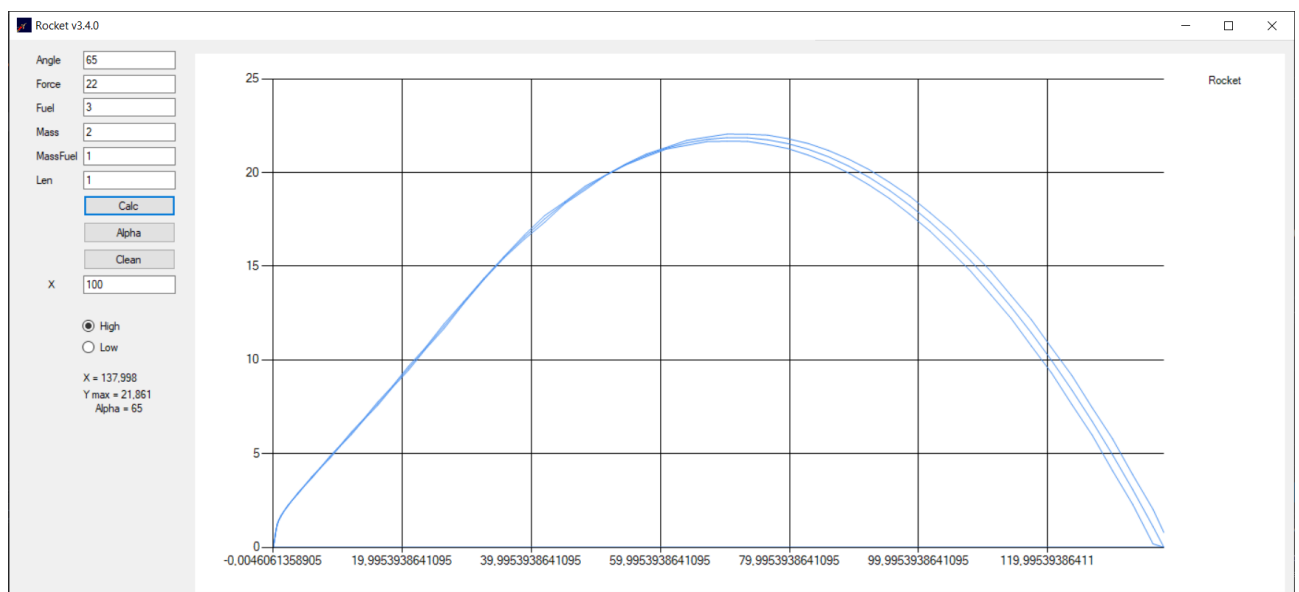


Рисунок 8 – Туннель погрешности

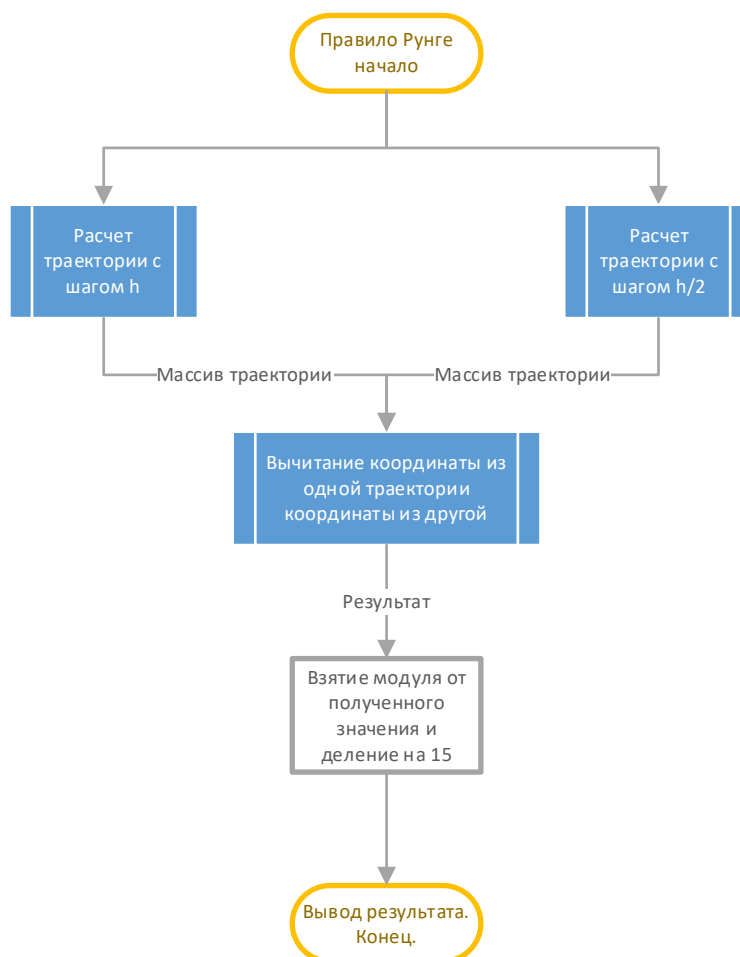


Рисунок 9 - Блок-схема правила Рунге

Класс Phase содержит в себе функции перегрузки операторов, для арифметических операций с объектами. Используется для реализации метода Рунге Кутта (рисунок 9). Объект класса phase имеет скорость по оси X и Y и координаты по оси Y и X.

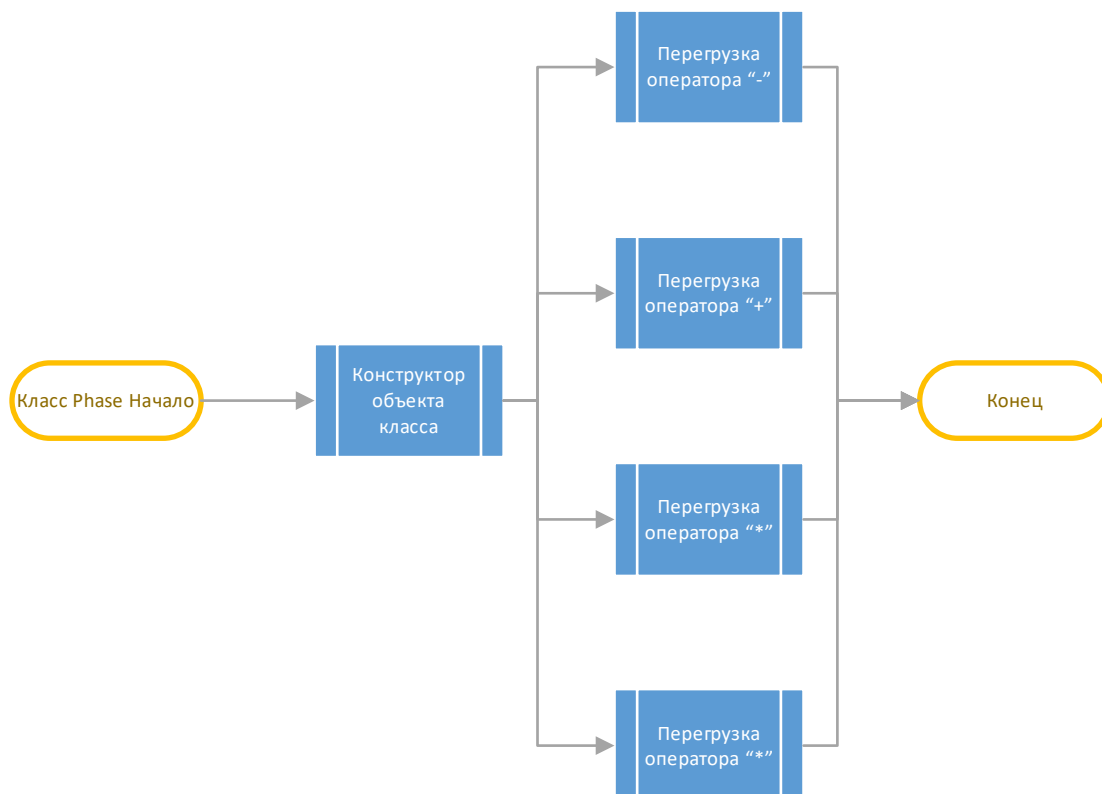


Рисунок 10 - Блок-схема класса Phase

К классу Model (рисунок 11) подключен класс Phase. Класс Model содержит в себе функции для расчета траектории ракеты по направляющей и свободное падение. Для математических функций подключена библиотека math.h.

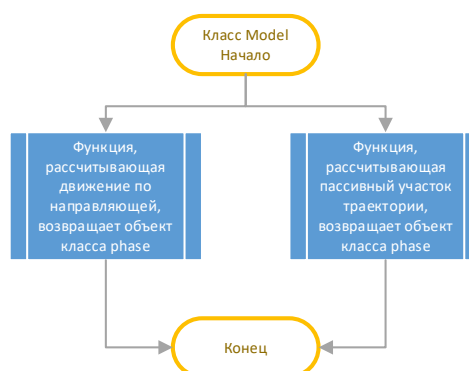


Рисунок 11- Блок-схема класса Model



К классу Numeric (рисунок 12) подключен класс Model. Содержит в себе расчет коэффициентов и реализацию метода Рунге-Кутты. Для расчета используются объекты и функции, описанные в других классах.

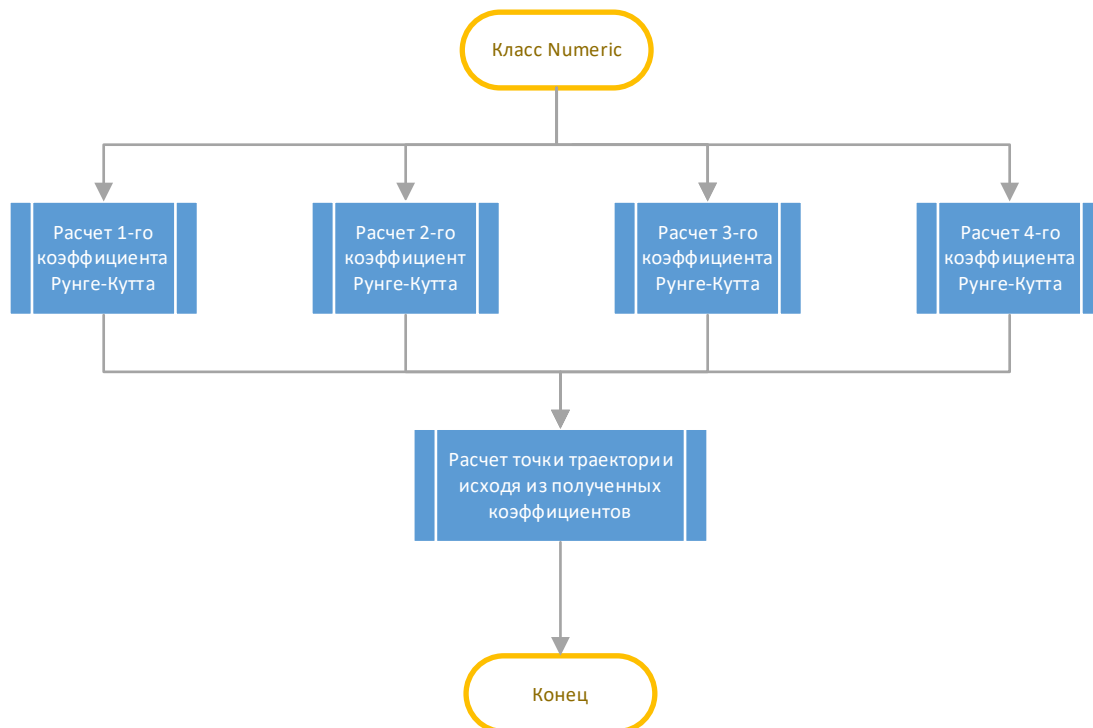


Рисунок 12 – Блок-схема класса Numeric

К классу Counting (рисунок 13) подключен класс numeric, подключаются библиотеки string и vector. В этом классе происходит расчет всей траектории ракеты. Для расчета используются методы, описанные в вышеупомянутых классах, в которые передаются значения для расчета, полученные из текстовых полей на форме.

:

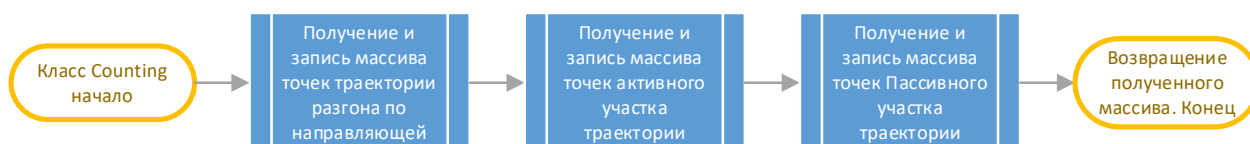


Рисунок 13 - Блок-схема класса Counting

Файл MyForm (рисунок 14) является точкой входа в программу. Здесь элементы (кнопки, текст-боксы, лейблы и так далее) наносятся на форму (окно программы), отрабатываются нажатия на кнопку Calc и Alpha.

Если пользователь нажал на кнопку Calc, класс Counting принимает значения из текст-боксов и рассчитывает траекторию полета ракеты, возвращает полученный массив, который принимает объект chart и выводит визуализированную траекторию. Далее программа выводит угол запуска (значение из текст-бокса Angle) максимальной высоты полета, и дальности (Из массива, который возвращает метод класса counting).

Если пользователь нажал на кнопку Alpha, программа запускает функцию золотого сечения, которая находит угол, при котором ракета улетит на наибольшее расстояние, далее запускает проверку, в которой сравнивает дальность, введенную пользователем с максимальной, в случае если дальность, введенная пользователем, превышает максимальную, программа выводит сообщение об ошибке, в котором содержится максимальная дальность.

Для нахождения угла, при котором ракета улетит на заданное пользователем расстояние, программа использует для расчета интервал углов от 0 до 90 так как для полета на дистанцию меньше максимальной ракета имеет два угла запуска, то, чтобы разграничить эти углы в программу был добавлен переключатель high/low.

Если переключатель стоит в положении high программа рассматривает интервал от угла при полете на наибольшее расстояние до 90. Далее программа запускает метод деления пополам, с помощью этого метода находит угол для полета на заданное расстояние.

Если переключатель стоит в положении Low, то программа рассматривает интервал от 0 до угла при полете на наибольшее расстояние и выполняет действия, описанные выше.

Далее программа выводит угол запуска (полученный в результате работы метода деления пополам) максимальной высоты полета, и дальности (Из массива, который возвращает метод класса counting).

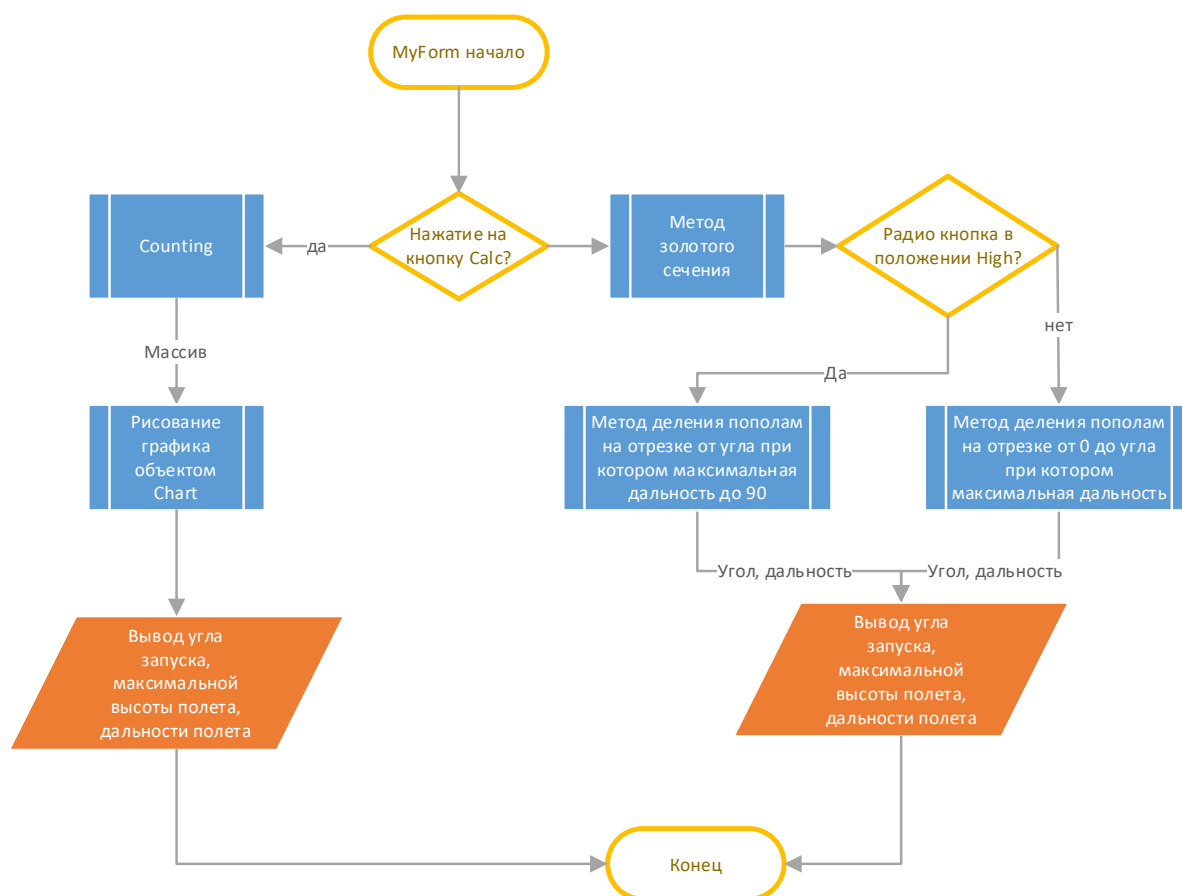


Рисунок 14 - Блок-схема MyForm

## 1.6 Предложения по практическому применению

С помощью программы и аналитических способностях пользователя возможно анализировать подбирать материалы для ракеты через ее вес, вес топлива, радиус полета и силу тяги двигателя для того, чтобы ракета преодолела заданное расстояние.

Таким образом, при разработке ракеты, после получения технического задания, можно использовать эту программу для подбора характеристик. Такой подход ускоряет стадию разработки. С помощью этой программы пользователь может снизить количество экспериментальных запусков до минимума.

Так же программу можно использовать для уже созданных ракет.

Ракеты могут доставлять боевые заряды и медикаменты на поле битвы или труднодоступные места, а приложение рассчитывать для них траекторию.

### 1.7 Перспективы дальнейшей разработки

В будущем планируется добавить в программу силу сопротивления воздуха, изменение гравитации в зависимости от высоты, это сделает траекторию, полученную с помощью программы сильно приближенной к реальной.

Учитывать шарообразность земли необходимо при полете на длинные дистанции. Это поможет уменьшить погрешность.

Добавить возможность моделировать полет для многоступенчатых ракет, это увеличит круг моделируемых программой объектов.

Рассматривать ракету как тело с геометрическими характеристиками необходимо, чтобы добавить в программу сопротивление воздуха, которое из-за больших скоростей сильно влияет на траекторию ракеты.

Добавить базу материалов, в которые можно добавлять свои, что бы программа выводила пользователю возможные материалы для строительства ракеты. Это введение должно сильно ускорить процесс разработки ракеты.

## 2 Программы - аналоги

Полных аналогов найдено не было. Существует несколько программ, которые позволяют получать пользователю некоторые данные полета ракеты.

### 2.1 LANE

Программа (рисунок 15), предназначена для моделирования траекторий движения различных летательных аппаратов (планеров, аэростатов, ракет, спускаемых аппаратов, парашютов и так далее) и баллистических тел (пуль, мячей, снарядов, метеоритов и так далее). Однако эту программу можно использовать исключительно в учебных некоммерческих целях. Программа на вход получает множество данных. Имеет сложный интерфейс. Программа имеет закрытый код, следовательно, модернизация программы под собственные нужды. Приложение является бесплатным.

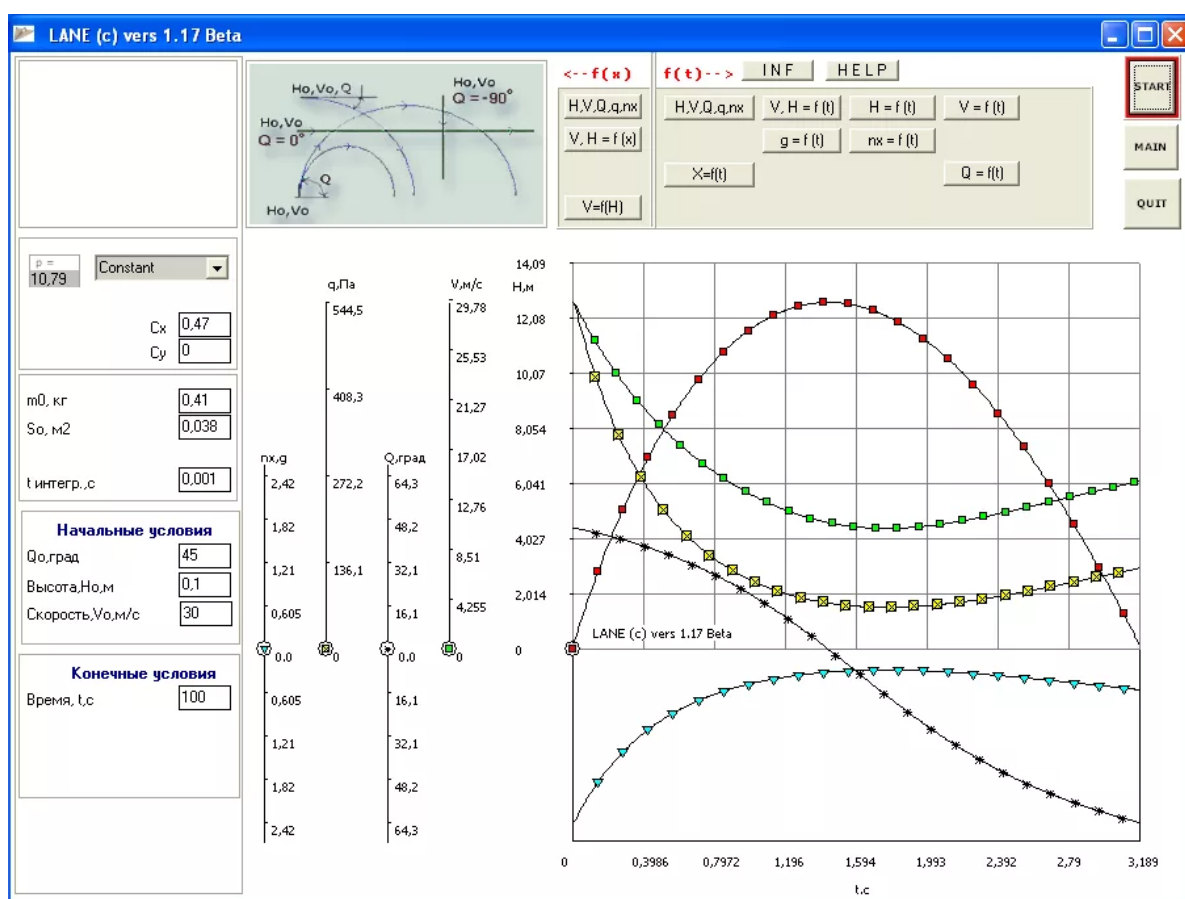


Рисунок 15 – Интерфейс программы LANE

## 2.2 Open rocket

Программа предназначена для любительского ракетостроения, позволяет начертить 3д модель ракеты, вычислить высоту подъема ракеты, имеет базу материалов и двигателей, однако в программе нет возможности смоделировать полет ракеты при запуске под углом отличным от вертикального. Приложение является бесплатным.

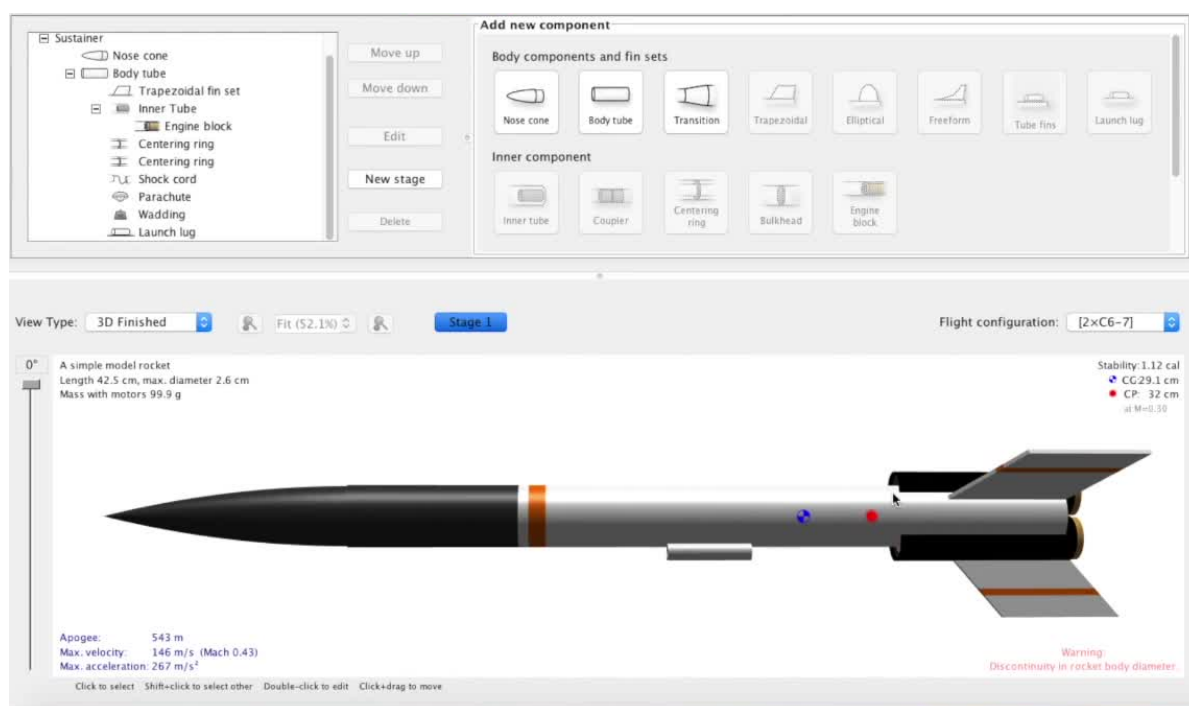


Рисунок 16 – Интерфейс программы Open rocket

### 3 Заключение

В итоге проделанной работы получены следующие результаты:

1. Создано приложение с визуализацией траектории ракеты
2. Есть возможность задавать физические характеристики для ракеты
3. Есть возможность рассчитать угол полета ракеты на расстояние, которое позволяют характеристики ракеты.
4. Программа может выводить несколько траекторий в поле графика.
5. У программы простой интерфейс (рисунки 17 - 19) и немного входных данных, что делает работу в ней быстрой и комфортной.

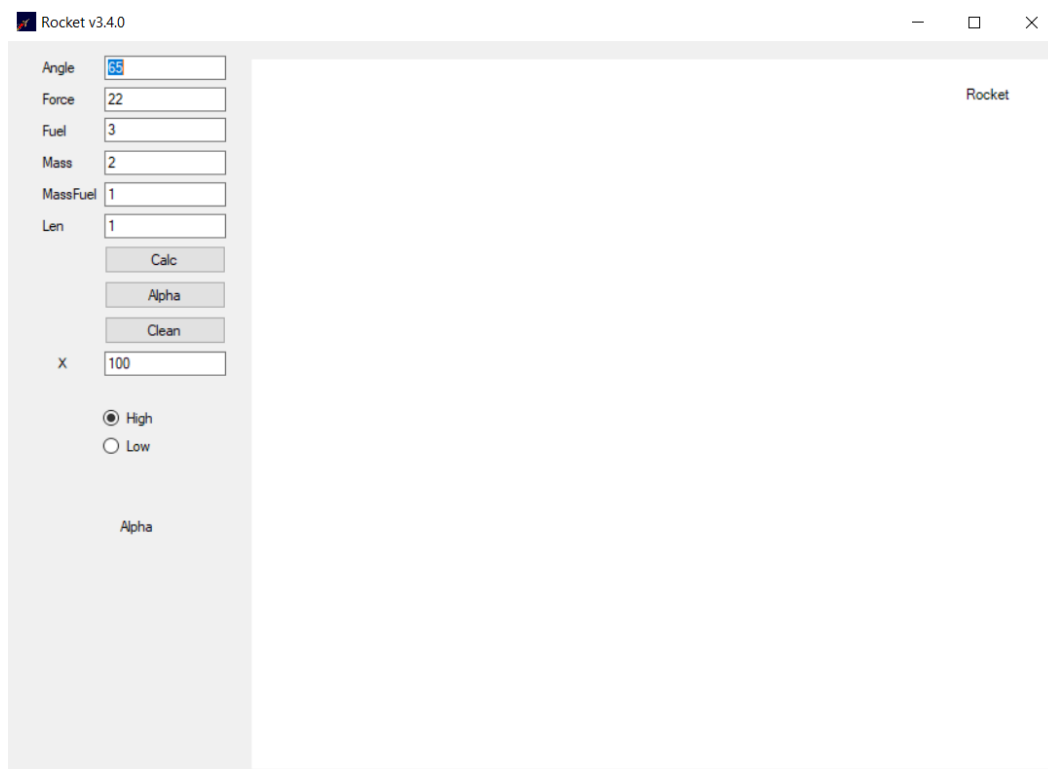


Рисунок 17 – Интерфейс программы (после запуска)

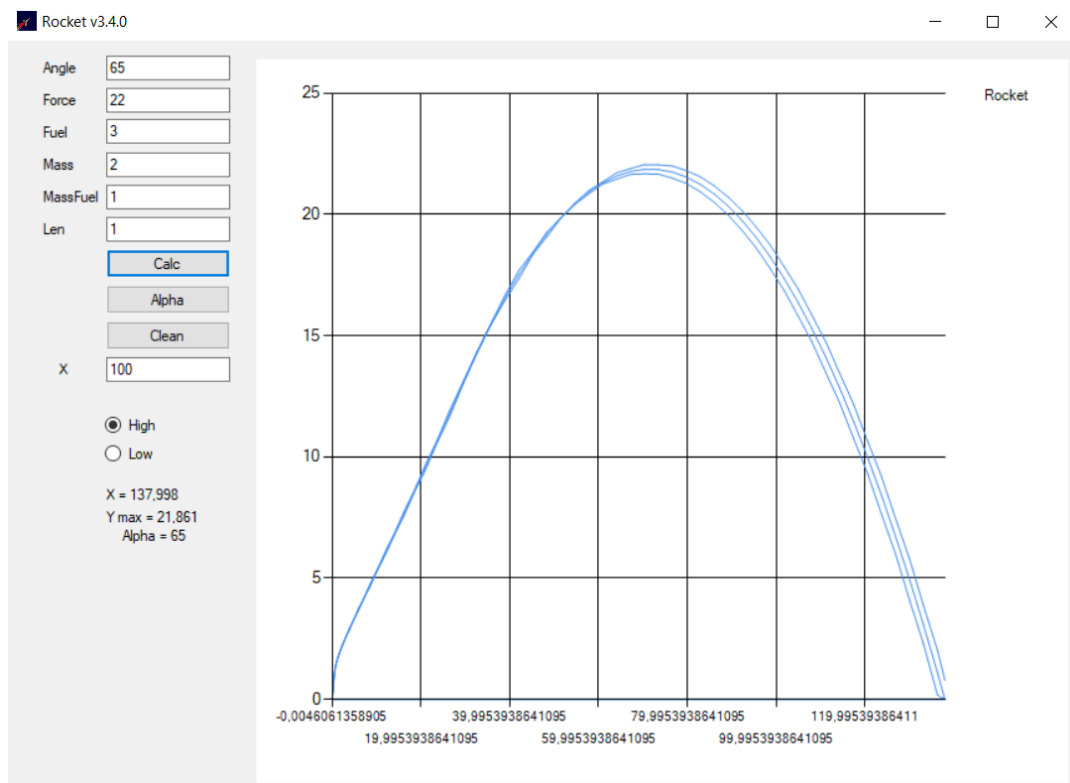


Рисунок 18 – Интерфейс программы (после ввода данных и нажатия на кнопку Calc)



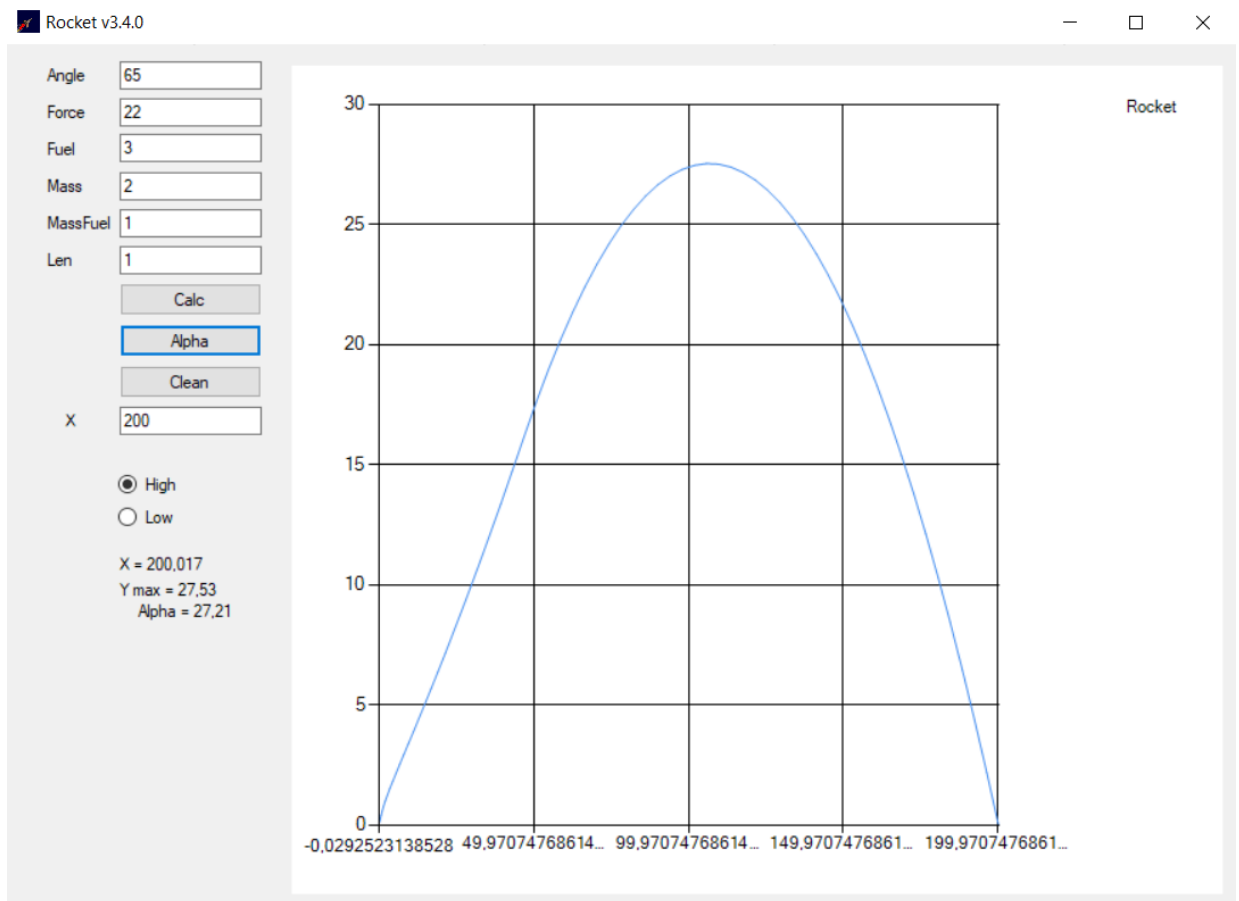


Рисунок 19 – Интерфейс программы (после нажатия на кнопку Alpha)

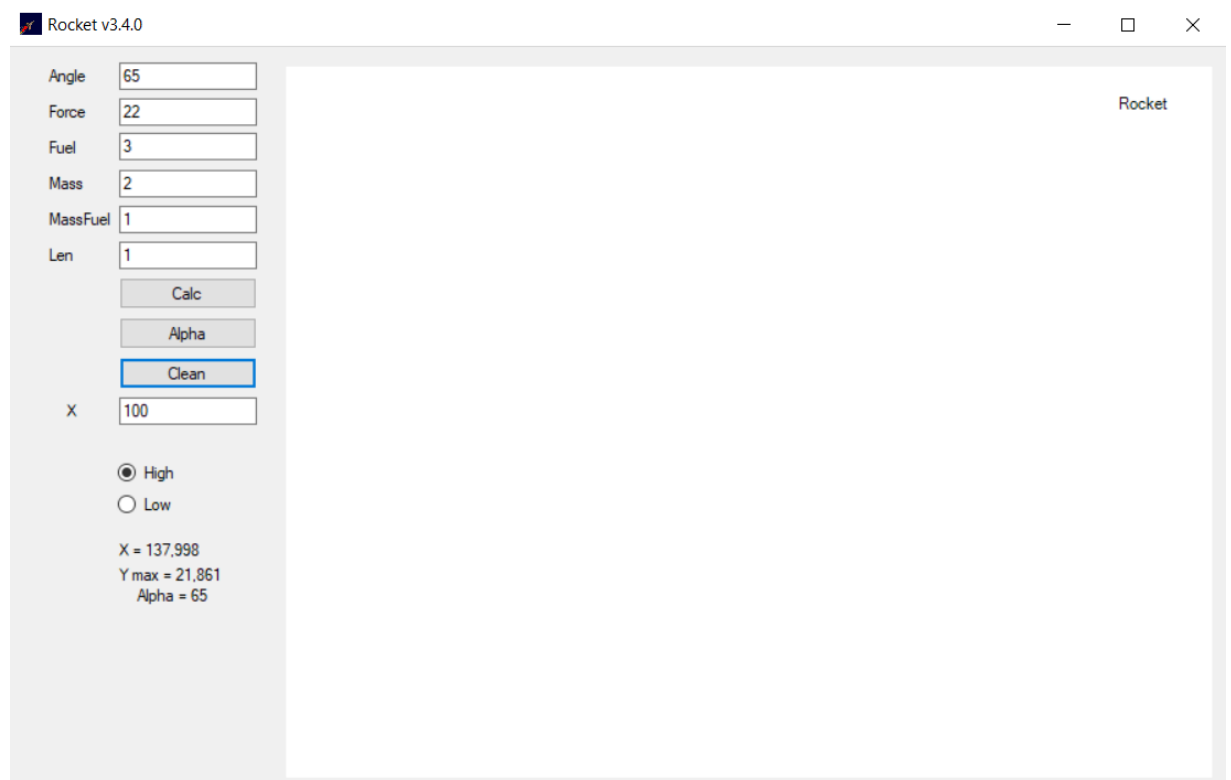


Рисунок 19 – Интерфейс программы (после нажатия на кнопку Clean)

#### 4 Список использованных источников

1. Материалы конференции “Шаг в будущее” прошлых лет. URL: <https://olymp.bmstu.ru/ru/publication> (Дата обращения 15.11.2020).
2. Сайт программы Open rocket. URL: [www.openrocket.info](http://www.openrocket.info) (Дата обращения 10.06.2020).
3. Сайт программы LANE. URL: [www.rakov.de](http://www.rakov.de) (Дата обращения 10.06.2020).
4. Сайт открытого сообщества программистов StackOverflow URL: [www.stackoverflow.com](http://www.stackoverflow.com) (Дата обращения 17.02.2021).

## Приложение А. Листинг программ

### Код файла phase.cpp

```
#include "phase.h"
#include <math.h>

phase::phase(double vx, double vy, double x, double y)
{
    this->vx = vx;
    this->vy = vy;
    this->x = x;
    this->y = y;
}

phase phase::absp(phase p)
{
    return phase(fabs(p.vx), fabs(p.vy), fabs(p.x), fabs(p.y));
}

phase phase::operator+(phase p)
{
    return phase(vx + p.vx, vy + p.vy, x + p.x, y + p.y);
}

phase phase::operator-(phase p)
{
    return phase(vx - p.vx, vy - p.vy, x - p.x, y - p.y);
}

phase phase::operator*(double k)
{

```

```
    return phase(vx * k, vy * k, x * k, y * k);  
}
```

```
phase phase::operator/(double k)  
{  
    return phase(vx / k, vy / k, x / k, y / k);  
}
```

```
#pragma once

class phase
{
public:
    double vx, vy, x, y;
    phase(double vx, double vy, double x, double y);
    static phase absp(phase p);
    // commutative
    phase operator+(phase p);
    phase operator-(phase p);
    // not commutative
    phase operator*(double k);
    phase operator/(double k);
};
```

```
#include "numeric.h"

double numeric::h = 0;

phase numeric::k1(phase p, double t)
{
    return model::df(p, t);
}

phase numeric::k2(phase p, double t, phase k1)
{
    return model::df(p + k1 * h / 2, t + h / 2);
}

phase numeric::k3(phase p, double t, phase k2)
{
    return model::df(p + k2 * h / 2, t + h / 2);
}

phase numeric::k4(phase p, double t, phase k3)
{
    return model::df(p + k3 * h, t + h);
}

phase numeric::next(phase prev, double t)
{
    phase _k1 = k1(prev, t);
    phase _k2 = k2(prev, t, _k1);
    phase _k3 = k3(prev, t, _k2);
```

```

    phase_k4 = k4(prev, t, _k3);

    return prev + (_k1 + _k2 * 2 + _k3 * 2 + _k4) * h / 6;
}

bool numeric::ground_check(phase& pb, phase& pa)
{
    if (pb.y < 0)
    {
        double endx = -pa.y * (pb.x - pa.x) / (pb.y - pa.y) + pa.x;
        // скорость по хорошему тоже пересчитать
        pb = phase(pa.vx, pa.vy, endx, 0);
        return true;
    }
    return false;
}

```

## Код файла numeric.h

```
#pragma once
#include "model.h"

class numeric
{
public:
    static double h;

    static phase k1(phase p, double t);
    static phase k2(phase p, double t, phase k1);
    static phase k3(phase p, double t, phase k2);
    static phase k4(phase p, double t, phase k3);

    static phase next(phase prev, double t);
    static bool ground_check(phase& pb, phase& pa);
};
```



## Код файла model.cpp

```
#include "model.h"

double model::mass0, model::rate, model::force, model::g, model::angle = 0;
phase model::p0_free = phase(0, 0, 0, 0);
double model::t0_free = 0;

double model::vel(double vx, double vy)
{
    return sqrt(vx * vx + vy * vy);
}

double model::mass(double t)
{
    return mass0 - rate * t;
}

//производные от скорости и координат по времени
phase model::df(phase p, double t)
{
    double dvx = (p.vx * force) / mass(t) / vel(p.vx, p.vy);
    double dvy = (p.vy * force) / mass(t) / vel(p.vx, p.vy) - g;
    double dx = p.vx;
    double dy = p.vy;

    return phase(dvx, dvy, dx, dy);
}

// guide - направляющая
double model::s_guide(double t)
```

```

{
    return force / rate * (t + (t - mass0 / rate) * log(mass0 / mass(t)))
        - g * sin(angle) * pow(t, 2) / 2;
}

double model::v_guide(double t)
{
    return force / rate * log(mass0 / mass(t)) - g * sin(angle) * t;
}

phase model::p_guide(double t)
{
    return phase(v_guide(t) * cos(angle), v_guide(t) * sin(angle),
        s_guide(t) * cos(angle), s_guide(t) * sin(angle));
}

// free - свободное падение
phase model::p_free(double t)
{
    double tau = t - t0_free;
    double vx = p0_free.vx;
    double vy = p0_free.vy - g * tau;
    double x = p0_free.x + p0_free.vx * tau;
    double y = p0_free.y + p0_free.vy * tau - g * tau * tau / 2;

    return phase(vx, vy, x, y);
}

```

## Код файла model.h

```
#pragma once
#include<math.h>
#include"phase.h"

class model
{
public:
    static double mass0, rate, force, g, angle;
    static double vel(double vx, double vy);
    static double mass(double t);
    // {vx, vy, x, y}
    static phase df(phase p, double t);

    // guide - направляющая
    static double s_guide(double t);
    static double v_guide(double t);
    static phase p_guide(double t);

    // free - свободное падение
    static phase p0_free;
    static double t0_free;
    static phase p_free(double t);
};
```

#### Код файла inverse.cpp

```
#include "inverse.h"

inverse::inverse(double x, double y, double angle)
{
    this->x = x;
    this->y = y;
    this->angle = angle;
}
```

#### Код файла inverse.h

```
#pragma once
class inverse
{
public:
    double x, y, angle;
    inverse(double x, double y, double angle);
};
```

## Код файла counting.cpp

```
#pragma once
#include "counting.h"

//Конструктор класса counting в котором передаются значения из текстовых.
counting::counting(String^ angle, String^ force, String^ len, String^ fuel_time, String^
mass, String^ massfuel)
{
    this->angle = Convert::ToDouble(angle);
    this->force = Convert::ToDouble(force);
    this->len = Convert::ToDouble(len);
    this->fuel_time = Convert::ToDouble(fuel_time);
    this->mass = Convert::ToDouble(mass);
    this->massfuel = Convert::ToDouble(massfuel);
}

//Метод класса counting который возвращает тип vector. Метод возвращает
координаты всех точек на траектории
vector<phase> counting::Coordinates(double angle, double step) {

    // Проверка на то что программа начинает исполнение с нажатия на кнопку
    calck

    if (angle < 0)
        angle = this->angle;

    //Перевод угла из градусов в радианы
    angle *= (M_PI) / 180;

    // инициализация параметров модели
    model::mass0 = mass;

    //находим расход топлива за секунду (кг)
```

```

model::rate = massfuel / fuel_time;
model::force = force;
model::g = g;
model::angle = angle;
if (step < 0)
    step = dt;
numeric::h = step;

vector<phase> p;
p.push_back(phase(0, 0, 0, 0));
double t = 0;

// разгон по направляющей
while (p.back().x <= len * cos(angle))
{
    p.push_back(model::p_guide(t));
    t += step;
}

//находим траекторию подъема на двигателе
while (model::rate * t <= massfuel)
{
    p.push_back(numeric::next(p.back(), t));

    //проверка на то что ракета не врезалась в землю
    if (numeric::ground_check(p.back(), p[p.size() - 2]))
        break;

    t += step;
}

```

```

// инициализация начальных условий свободного падения
model::p0_free = p.back();
model::t0_free = t;

//траектория при отработавшем двигателе
while (!numeric::ground_check(p.back(), p[p.size() - 2]))
{
    p.push_back(model::p_free(t));
    t += step;
}

return p;
}

vector<phase> counting::error(double angle)
{
    if (angle < 0)
        angle = this->angle;

    //Перевод угла из градусов в радианы
    vector<phase> p_h = Coordinates(angle);
    vector<phase> p_2h = Coordinates(angle, 2 * dt);

    vector<phase> err;// = vector<phase>();
    for (int i = 0; i < p_h.size(); i++)
    {
        int j = i / 2;
        err.push_back(phase::absp((p_2h[j] - p_h[i]) / 15));
    }
}

```

```

        return err;
    }

double counting::flyLenght(double angle)
{
    return Coordinates(angle).back().x;
}

//МЕТОД ЗОЛОТОГО СЕЧЕНИЯ
inverse counting::findMax()
{
    double a = 0, b = 90;
    double x1 = b - (b - a) / phi;
    double x2 = a + (b - a) / phi;
    double y1 = flyLenght(x1);
    double y2 = flyLenght(x2);
    while (b - a > prec)
    {
        if (y1 <= y2)
        {
            a = x1;
            x1 = x2; y1 = y2;
            x2 = a + (b - a) / phi;
            y2 = flyLenght(x2);
        }
        else
        {
            b = x2;
            x2 = x1; y2 = y1;
            x1 = b - (b - a) / phi;

```



```

        y1 = flyLenght(x1);
    }
}

double alpha_max = (a + b) / 2;
return inverse(flyLenght(alpha_max), 0, alpha_max);
}

//метод деления пополам
inverse counting::findAlphaInv(double x, double alpha_max, bool high)
{
    double a = 0, b = 90;
    if (high)
        a = alpha_max;
    else
        b = alpha_max;

    double mid = (a + b) / 2;

    //double diff_a = flyLenght(a) - x;
    double diff_b = flyLenght(b) - x;
    double diff_mid = flyLenght(mid) - x;
    while (b - a > prec)
    {
        if (diff_b * diff_mid < 0)
            a = mid;
        else
        {
            b = mid;
            diff_b = flyLenght(b) - x;
        }
    }
}

```

```

        mid = (a + b) / 2;
        diff_mid = flyLenght(mid) - x;
    }

    vector<phase> p = Coordinates(mid);

    double y_max = 0;
    for (int i = 0; i < p.size(); i++)
        if (p[i].y > y_max)
            y_max = p[i].y;

    return inverse(p.back().x, y_max, mid);
}

```

## Код файла counting.h

```
#pragma once
#include<string>
#include<vector>
#define _USE_MATH_DEFINES
#include "numeric.h"
#include "inverse.h"
using namespace std;
using namespace System;

class counting
{
public:
    const double g = 9.8;
    const double dt = 0.1; // интервал в секундах с которым программа считает
    траекторию
    const double dt2d2 = dt * dt / 2;
    const double prec = 0.1; // точность для обратной задачи
    const double phi = (1.0 + sqrt(5)) / 2;

    double angle, force, len, fuel_time, mass, massfuel;
    int i;

    counting(String^ angle, String^ force, String^ len, String^ fuel, String^ mass,
    String^ massfuel);
    vector<phase> Coordinates(double angle = -1, double step = -1);

    vector<phase> error(double angle = -1);

    double flyLenght(double angle);
```

```
inverse findMax();  
inverse findAlphaInv(double x, double alpha_max, bool high);  
};
```

#### Код файла MyForm.cpp

```
//Файл для подключения конструктора WinApi

#include <Windows.h>
#include "MyForm.h"
using namespace Rocket;
int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Application::Run(gnew MyForm);
    return 0;
}
```

#### Код файла MyForm.h

```
#pragma once
#include "counting.h"
namespace Rocket {
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    #pragma region Windows Form Designer generated code
    /// <summary>
    /// Сводка для MyForm
    /// </summary>
    public ref class MyForm : public System::Windows::Forms::Form
    {
    }
```

public:

```
MyForm(void)
{
    InitializeComponent();
    //
    //TODO: добавьте код конструктора
    //
}
```

protected:

```
/// <summary>
/// Освободить все используемые ресурсы.
/// </summary>
~MyForm()
{
    if (components)
    {
        delete components;
    }
}
```

private: System::Windows::Forms::Label^ label6;

protected:

private: System::Windows::Forms::TextBox^ textBoxLen;

private: System::Windows::Forms::Label^ labelYmax;

private: System::Windows::Forms::Label^ labelXmax;

private: System::Windows::Forms::RadioButton^ radioButtonLow;

private: System::Windows::Forms::RadioButton^ radioButtonHigh;

private: System::Windows::Forms::Label^ labelAlpha;

private: System::Windows::Forms::Label^ label5;

private: System::Windows::Forms::TextBox^ textBoxX;

```

private: System::Windows::Forms::Button^ buttonAlpha;

private: System::Windows::Forms::Button^ buttonCalc;
private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::TextBox^ textBoxMass;
private: System::Windows::Forms::TextBox^ textBoxFuel;
private: System::Windows::Forms::TextBox^ textBoxForce;
private: System::Windows::Forms::TextBox^ textBoxAngle;
private: System::Windows::Forms::DataVisualization::Charting::Chart^ chart1;
private: System::Windows::Forms::Label^ label7;
private: System::Windows::Forms::TextBox^ textBoxMassFuel;
private: System::Windows::Forms::Button^ ButtonClean;

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container^ components;

    /// <summary>
    /// Требуемый метод для поддержки конструктора — не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void)
    {

        System::Windows::Forms::DataVisualization::Charting::ChartArea^

```

```

chartArea1                                     =                               (gcnew
System::Windows::Forms::DataVisualization::Charting::ChartArea());
        System::Windows::Forms::DataVisualization::Charting::Legend^
legend1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Legend());

        System::Windows::Forms::DataVisualization::Charting::LegendCellColumn^
legendCellColumn1                             =                               (gcnew
System::Windows::Forms::DataVisualization::Charting::LegendCellColumn());
        System::Windows::Forms::DataVisualization::Charting::Series^
series1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Series());
        System::Windows::Forms::DataVisualization::Charting::Title^
title1 = (gcnew System::Windows::Forms::DataVisualization::Charting::Title());
        System::ComponentModel::ComponentResourceManager^
resources                                     =                               (gcnew
System::ComponentModel::ComponentResourceManager(MyForm::typeid));
        this->label6 = (gcnew System::Windows::Forms::Label());
        this->textBoxLen                     =                               (gcnew
System::Windows::Forms::TextBox());
        this->labelYmax = (gcnew System::Windows::Forms::Label());
        this->labelXmax = (gcnew System::Windows::Forms::Label());
        this->radioButtonLow                 =                               (gcnew
System::Windows::Forms::RadioButton());
        this->radioButtonHigh                =                               (gcnew
System::Windows::Forms::RadioButton());
        this->labelAlpha = (gcnew System::Windows::Forms::Label());
        this->label5 = (gcnew System::Windows::Forms::Label());
        this->textBoxX = (gcnew System::Windows::Forms::TextBox());
        this->buttonAlpha = (gcnew System::Windows::Forms::Button());
        this->buttonCalc = (gcnew System::Windows::Forms::Button());
        this->label4 = (gcnew System::Windows::Forms::Label());

```



```

        this->label3 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->textBoxMass = (gcnew
System::Windows::Forms::TextBox());
        this->textBoxFuel = (gcnew
System::Windows::Forms::TextBox());
        this->textBoxForce = (gcnew
System::Windows::Forms::TextBox());
        this->textBoxAngle = (gcnew
System::Windows::Forms::TextBox());
        this->chart1 = (gcnew
System::Windows::Forms::DataVisualization::Charting::Chart());
        this->label7 = (gcnew System::Windows::Forms::Label());
        this->textBoxMassFuel = (gcnew
System::Windows::Forms::TextBox());
        this->ButtonClean = (gcnew System::Windows::Forms::Button());

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))-
>BeginInit();

        this->SuspendLayout();
        //
        // label6
        //
        this->label6->AutoSize = true;
        this->label6->Location = System::Drawing::Point(26, 145);
        this->label6->Name = L"label6";
        this->label6->Size = System::Drawing::Size(28, 15);
        this->label6->TabIndex = 41;
        this->label6->Text = L"Len";

```

```

//
// textBoxLen
//
this->textBoxLen->Location = System::Drawing::Point(80, 142);
this->textBoxLen->Name = L"textBoxLen";
this->textBoxLen->Size = System::Drawing::Size(100, 20);
this->textBoxLen->TabIndex = 40;
this->textBoxLen->Text = L"1";
//
// labelYmax
//
this->labelYmax->AutoSize = true;
this->labelYmax->Location = System::Drawing::Point(77, 377);
this->labelYmax->Name = L"labelYmax";
this->labelYmax->Size = System::Drawing::Size(0, 15);
this->labelYmax->TabIndex = 39;
//
// labelXmax
//
this->labelXmax->AutoSize = true;
this->labelXmax->Location = System::Drawing::Point(77, 359);
this->labelXmax->Name = L"labelXmax";
this->labelXmax->Size = System::Drawing::Size(0, 15);
this->labelXmax->TabIndex = 38;
//
// radioButtonLow
//
this->radioButtonLow->AutoSize = true;
this->radioButtonLow->Location = System::Drawing::Point(79,
324);

```

```

this->radioButtonLow->Name = L"radioButtonLow";
this->radioButtonLow->Size = System::Drawing::Size(51, 19);
this->radioButtonLow->TabIndex = 37;
this->radioButtonLow->Text = L"Low";
this->radioButtonLow->UseVisualStyleBackColor = true;
//
// radioButtonHigh
//
this->radioButtonHigh->AutoSize = true;
this->radioButtonHigh->Checked = true;
this->radioButtonHigh->Location = System::Drawing::Point(79,
301);

this->radioButtonHigh->Name = L"radioButtonHigh";
this->radioButtonHigh->Size = System::Drawing::Size(54, 19);
this->radioButtonHigh->TabIndex = 36;
this->radioButtonHigh->TabStop = true;
this->radioButtonHigh->Text = L"High";
this->radioButtonHigh->UseVisualStyleBackColor = true;
//
// labelAlpha
//
this->labelAlpha->AutoSize = true;
this->labelAlpha->Location = System::Drawing::Point(90, 392);
this->labelAlpha->Name = L"labelAlpha";
this->labelAlpha->Size = System::Drawing::Size(38, 15);
this->labelAlpha->TabIndex = 35;
this->labelAlpha->Text = L"Alpha";
//
// label5
//

```

```

this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(39, 258);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(15, 15);
this->label5->TabIndex = 34;
this->label5->Text = L"X";
//
// textBoxX
//
this->textBoxX->Location = System::Drawing::Point(80, 255);
this->textBoxX->Name = L"textBoxX";
this->textBoxX->Size = System::Drawing::Size(100, 20);
this->textBoxX->TabIndex = 33;
this->textBoxX->Text = L"100";
//
// buttonAlpha
//
this->buttonAlpha->Location = System::Drawing::Point(80, 197);
this->buttonAlpha->Name = L"buttonAlpha";
this->buttonAlpha->Size = System::Drawing::Size(100, 23);
this->buttonAlpha->TabIndex = 32;
this->buttonAlpha->Text = L"Alpha";
this->buttonAlpha->UseVisualStyleBackColor = true;
this->buttonAlpha->Click += gcnew System::EventHandler(this,
&MyForm::buttonAlpha_Click);
//
// buttonCalc
//
this->buttonCalc->Location = System::Drawing::Point(80, 168);
this->buttonCalc->Name = L"buttonCalc";

```

```

this->buttonCalc->Size = System::Drawing::Size(100, 23);
this->buttonCalc->TabIndex = 29;
this->buttonCalc->Text = L"Calc";
this->buttonCalc->UseVisualStyleBackColor = true;
this->buttonCalc->Click += gcnew System::EventHandler(this,
&MyForm::buttonCalc_Click);

//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(26, 93);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(37, 15);
this->label4->TabIndex = 28;
this->label4->Text = L"Mass";
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(26, 67);
this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(31, 15);
this->label3->TabIndex = 27;
this->label3->Text = L"Fuel";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(26, 41);
this->label2->Name = L"label2";

```

```

this->label2->Size = System::Drawing::Size(38, 15);
this->label2->TabIndex = 26;
this->label2->Text = L"Force";
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(26, 15);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(38, 15);
this->label1->TabIndex = 25;
this->label1->Text = L"Angle";
//
// textBoxMass
//
this->textBoxMass->Location = System::Drawing::Point(80, 90);
this->textBoxMass->Name = L"textBoxMass";
this->textBoxMass->Size = System::Drawing::Size(100, 20);
this->textBoxMass->TabIndex = 24;
this->textBoxMass->Text = L"2";
//
// textBoxFuel
//
this->textBoxFuel->Location = System::Drawing::Point(80, 63);
this->textBoxFuel->Name = L"textBoxFuel";
this->textBoxFuel->Size = System::Drawing::Size(100, 20);
this->textBoxFuel->TabIndex = 23;
this->textBoxFuel->Text = L"3";
//
// textBoxForce

```

```

//
this->textBoxForce->Location = System::Drawing::Point(80, 38);
this->textBoxForce->Name = L"textBoxForce";
this->textBoxForce->Size = System::Drawing::Size(100, 20);
this->textBoxForce->TabIndex = 22;
this->textBoxForce->Text = L"22";
//
// textBoxAngle
//
this->textBoxAngle->Location = System::Drawing::Point(80, 12);
this->textBoxAngle->Name = L"textBoxAngle";
this->textBoxAngle->Size = System::Drawing::Size(100, 20);
this->textBoxAngle->TabIndex = 21;
this->textBoxAngle->Text = L"65";
//
// chart1
//
this->chart1->Anchor
static_cast<System::Windows::Forms::AnchorStyles>((((System::Windows::Forms::
AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Bottom)
| System::Windows::Forms::AnchorStyles::Left)
| System::Windows::Forms::AnchorStyles::Right));
chartArea1->Name = L"ChartArea1";
this->chart1->ChartAreas->Add(chartArea1);
legendCellColumn1->Name = L"Column1";
legend1->CellColumns->Add(legendCellColumn1);
legend1->Name = L"Legend1";
this->chart1->Legends->Add(legend1);
this->chart1->Location = System::Drawing::Point(201, 15);
this->chart1->Name = L"chart1";

```

```

series1->ChartArea = L"ChartArea1";
series1->ChartType
System::Windows::Forms::DataVisualization::Charting::SeriesChartType::Line;
series1->Legend = L"Legend1";
series1->Name = L"Rocket";
this->chart1->Series->Add(series1);
this->chart1->Size = System::Drawing::Size(655, 583);
this->chart1->TabIndex = 40;
this->chart1->Text = L"chart1";
title1->Name = L"Title1";
title1->Position->Auto = false;
title1->Position->Height = 2;
title1->Position->Width = 100;
title1->Position->Y = 3;
this->chart1->Titles->Add(title1);
//
// label7
//
this->label7->AutoSize = true;
this->label7->Location = System::Drawing::Point(26, 119);
this->label7->Name = L"label7";
this->label7->Size = System::Drawing::Size(61, 15);
this->label7->TabIndex = 43;
this->label7->Text = L"MassFuel";
//
// textBoxMassFuel
//
this->textBoxMassFuel->Location = System::Drawing::Point(80,
116);

this->textBoxMassFuel->Name = L"textBoxMassFuel";

```



```

this->textBoxMassFuel->Size = System::Drawing::Size(100, 20);
this->textBoxMassFuel->TabIndex = 42;
this->textBoxMassFuel->Text = L"1";
//
// ButtonClean
//
this->ButtonClean->Location = System::Drawing::Point(80, 226);
this->ButtonClean->Name = L"ButtonClean";
this->ButtonClean->Size = System::Drawing::Size(100, 23);
this->ButtonClean->TabIndex = 44;
this->ButtonClean->Text = L"Clean";
this->ButtonClean->UseVisualStyleBackColor = true;
this->ButtonClean->Click += gnew System::EventHandler(this,
&MyForm::ButtonClean_Click);
//
// MyForm
//
this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
this->AutoScaleMode
System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(866, 607);
this->Controls->Add(this->ButtonClean);
this->Controls->Add(this->label7);
this->Controls->Add(this->textBoxMassFuel);
this->Controls->Add(this->chart1);
this->Controls->Add(this->label6);
this->Controls->Add(this->textBoxLen);
this->Controls->Add(this->labelYmax);
this->Controls->Add(this->labelXmax);
this->Controls->Add(this->radioButtonLow);

```

```

        this->Controls->Add(this->radioButtonHigh);
        this->Controls->Add(this->labelAlpha);
        this->Controls->Add(this->label5);
        this->Controls->Add(this->textBoxX);
        this->Controls->Add(this->buttonAlpha);
        this->Controls->Add(this->buttonCalc);
        this->Controls->Add(this->label4);
        this->Controls->Add(this->label3);
        this->Controls->Add(this->label2);
        this->Controls->Add(this->label1);
        this->Controls->Add(this->textBoxMass);
        this->Controls->Add(this->textBoxFuel);
        this->Controls->Add(this->textBoxForce);
        this->Controls->Add(this->textBoxAngle);
        this->Icon = (cli::safe_cast<System::Drawing::Icon^>(resources-
>GetObject(L"$this.Icon"))));
        this->Name = L"MyForm";
        this->Text = L"Rocket v3.4.0";

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->chart1))-
>EndInit();

        this->ResumeLayout(false);
        this->PerformLayout();

    }

#pragma endregion

    //Выполняется при нажатии на кнопку Calc
    private: System::Void buttonCalc_Click(System::Object^ sender,
System::EventArgs^ e) {
        //передача значений из текстовых

```

```
counting ee(textBoxAngle->Text, textBoxForce->Text,  
            textBoxLen->Text,   textBoxFuel->Text,   textBoxMass->Text,  
textBoxMassFuel->Text);
```

//Передача значений из класса counting метода Coordinates в вектор  
graph

```
vector<phase> graph = ee.Coordinates();
```

```
vector<phase> error = ee.error();
```

```
vector<phase> err_up;
```

```
vector<phase> err_down;
```

```
for (int i = 0; i < graph.size(); i++)
```

```
{
```

```
    err_up.push_back(graph[i] + error[i]);
```

```
    err_down.push_back(graph[i] - error[i]);
```

```
}
```

```
err_down.back().y = 0;
```

```
double size = graph.size();
```

//Вывод в окно Угла при котором ракета прилетит в заданную точку

```
labelAlpha->Text = "Alpha = " + (textBoxAngle->Text);
```

//Вывод в окно дальности полета

```
labelXmax->Text = "X = " + Math::Round(graph[graph.size() - 1].x,  
3).ToString();
```

//Переменная для сравнения максимального элемента в цикле

```
double ymax = 0;
```

//Цикл для нахождения максимального элемента в Vector

```

for each (phase var in graph)
    if (ymax < var.y)
        ymax = var.y;

//Вывод максимальной высоты при полете с округлением до
ТЫСЯЧНЫХ
labelYmax->Text = "Y max = " + Math::Round(ymax, 3).ToString();

//Рисование графика полета ракеты с помощью chart
for (int i = 0; i < size; i++)
{
    //Передача значений в объект типа chart
    chart1->Series["Rocket"]->Points->AddXY(graph[i].x,
graph[i].y); //Значение X Y
}
for (int i = 0; i < size; i++)
{
    //Передача значений в объект типа chart
    chart1->Series["Rocket"]->Points->AddXY(graph[i].x,
err_down[i].y); //Значение X Y
}
for (int i = 0; i < size; i++)
{
    //Передача значений в объект типа chart
    chart1->Series["Rocket"]->Points->AddXY(graph[i].x,
err_up[i].y); //Значение X Y
}
}

// Выполняется при нажатии на кнопку Alpha

```

```

private:    System::Void    buttonAlpha_Click(System::Object^    sender,
System::EventArgs^ e) {
    //Считывание значения из текстового X
    double findX = Convert::ToDouble(textBoxX->Text);

    //передача значений из текстовых
    counting ee(textBoxAngle->Text, textBoxForce->Text,
        textBoxLen->Text,    textBoxFuel->Text,    textBoxMass->Text,
textBoxMassFuel->Text);

    //Обработка ошибок с помощью messagebox

    inverse data_max = ee.findMax();
    if (findX > data_max.x)
    {
        System::Windows::Forms::MessageBox::Show("Заданная
дальность полета превышает максимальную. " +
            "Максимальная дальность полета составляет " +
Math::Round(data_max.x, 3).ToString(), "Ошибка!");
        return;
    }
    if (findX < 10) // разобраться
    {
        System::Windows::Forms::MessageBox::Show("при полете на
такую дистанцию вы будете уничтожены. ", "Ошибка!");
        return;
    }

    inverse    data_target    =    ee.findAlphaInv(findX,    data_max.angle,
radioButtonHigh->Checked);

```

```

        //Вывод в окно Угла при котором ракета прилетит в заданную точку
        labelAlpha->Text = "Alpha = " + (((int)(data_target.angle * 100)) /
100.0f).ToString();

        //Вывод в окно дальности полета
        labelXmax->Text = "X = " + Math::Round(data_target.x, 3).ToString();

        //Вывод максимальной высоты при полете с округлением до
ТЫСЯЧНЫХ
        labelYmax->Text = "Y max = " + Math::Round(data_target.y,
3).ToString();

        //Передача значений из класса counting метода Coordinates в вектор
graph
        vector<phase> graph = ee.Coordinates(data_target.angle);

        //Рисование графика полета ракеты с помощью chart
        double size = graph.size();
        for (int i = 0; i < size; i++)
            //Передача значений в объект типа chart
            chart1->Series["Rocket"]->Points->AddXY(graph[i].x,
graph[i].y); //Значение X Y
    }

    private: System::Void ButtonClean_Click(System::Object^ sender,
System::EventArgs^ e) {
        chart1->Series["Rocket"]->Points->Clear();
    }
};
}

```