**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**"Уфимский университет науки и технологий"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Математическое моделирование

**Отчет по лабораторной работе № 1**

**Тема:** «Компьютерное моделирование движения космических тел»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМ-457 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Галямова А.З. |  |  |  |
| Принял | Лукащук С.Ю. |  |  |  |

**Уфа 2024**

**Цель работы:** получить навык численного расчета траекторий движения космических тел под действием гравитационных сил.

**Задание на лабораторную работу**

**Задача I.** Рассматривается динамика трех разновеликих небесных тел: звезды, планеты и ее спутника. В качестве примера рассматривается Солнечная система. Масса Солнца . Параметры двух других тел выбираются в соответствии с индивидуальным номером варианта из таблицы.

1) Составить уравнения движения второго и третьего тела в системе отсчета, связанной с первым (самым массивным) телом. Предполагается, что движение всех тел происходит в одной плоскости.

2) Написать программу численного интегрирования составленных уравнений движения и построить траектории движения тел. В качестве начальных условий принять следующие: все тела находятся на одной прямой, вектора скоростей движения второго и третьего тела сонаправлены. Расстояния между первым и вторым, а также вторым и третьим телами приведены в таблице. Там же указаны значения начальных скоростей второго и третьего тела. Исследовать отклонение орбиты планеты и спутника от круговой с течением времени, а также характер изменения их модулей скорости

**Задача II.** На круговой орбите высотой Н второго тела находится космический корабль. В некоторый момент времени его двигатели включаются и работают в течение времени Т выводя корабль на новую орбиту, пересекающую орбиту третьего тела. Вектор тяги двигателя в любой момент времени направлен по касательной к траектории движения. Определить местоположение космического корабля на первоначальной орбите в момент включения двигателя из условия минимума массы топлива, необходимой для доставки на поверхность третьего тела полезного груза массой М0. Местоположение определяется относительно прямой, соединяющей центры второго и третьего тел. Масса корабля складывается из массы топлива, полностью выгорающего за время Т, массы конструкции (0.025 стартовой массы) и массы полезной нагрузки Ма. В конце активного участка траектории (через время 7) происходит отделение полезного груза, который движется далее только под действием гравитационных сил. Скорость полезного груза при достижении поверхности третьего тела не ограничивается.

**Практическая часть**

**Задача I.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № вар | Параметры второго тела | | | | Параметры третьего тела | | | |
| , км | , км | , млн. км | , км/с | , кг | , км | , тыс. км | , км/с |
| 3 |  | 6378 | 150 | 30 |  | 1737 | 384 | 1 |

В системе действует сила притяжения космических тел, которая подчиняется закону всемирного тяготения:

– гравитационная постоянная, – массы космических тел.

Уравнение движения:

Для системы планета-спутник уравнения движения, относительно Солнца будут иметь вид:

Далее составим систему ОДУ:

Решив данную задачу Коши с помощью библиотеки Boost(ODEINT) на С++, получаем следующие траектории небесных тел:

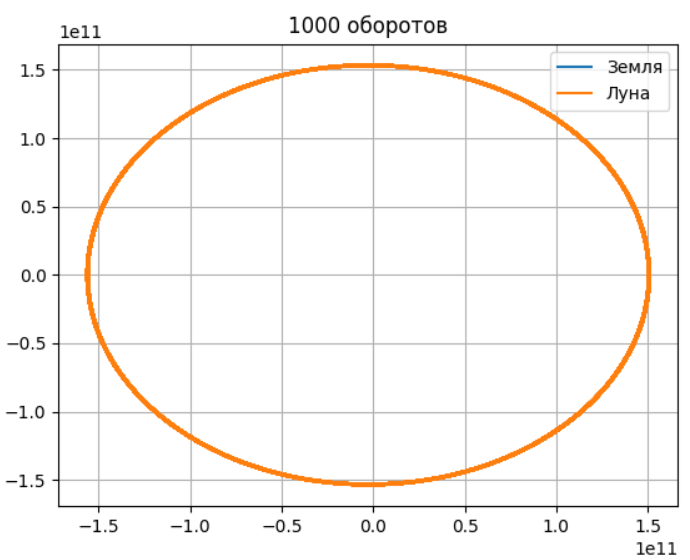


Рисунок 1 – Траектория движения планеты и спутника вокруг Солнца спустя 1000 оборотов

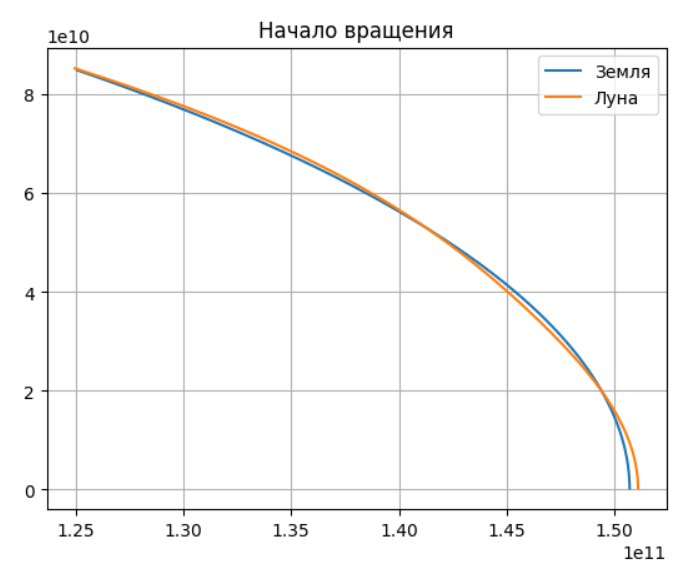


Рисунок 2 – Траектория движения планеты и спутника вокруг Солнца в начале движения

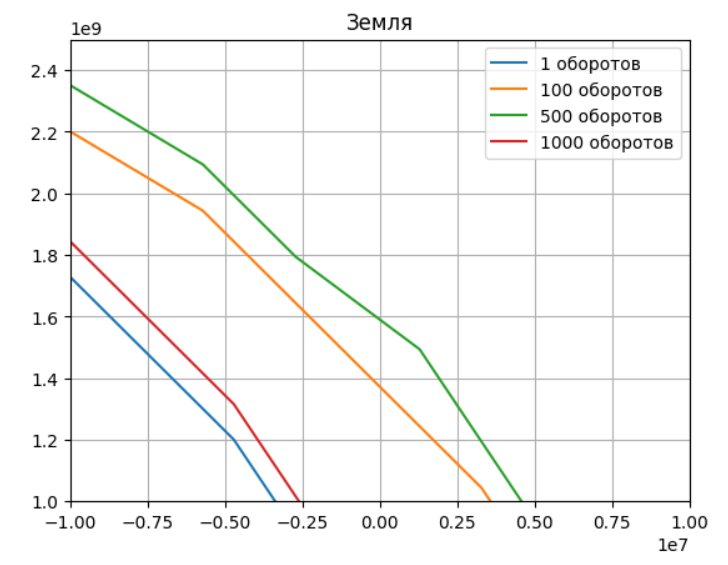


Рисунок 3 Орбита планеты после 1, 100, 500 и 1000 оборотов

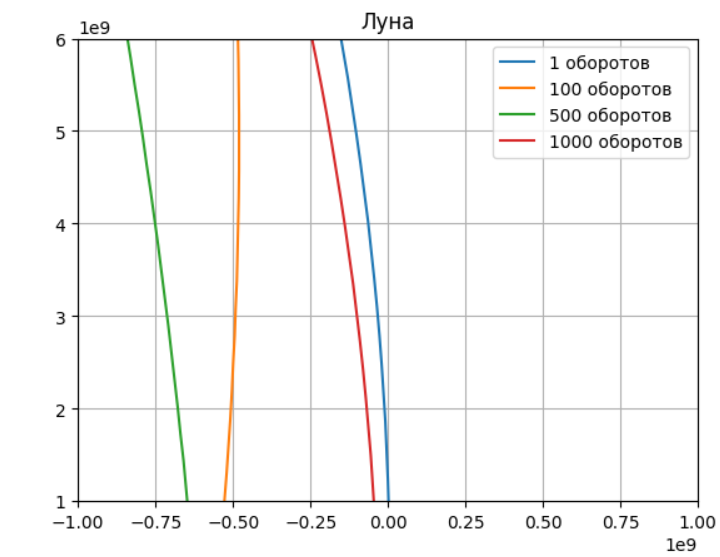


Рисунок 4 Орбита спутника после 1, 100, 500 и 1000 оборотов

**Задача II.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № варианта | H, км |  |  | Характеристики топлива | | |
| Горючее | Окислитель | Скорость истечения, м/с |
| 3 | 300 | 2400 | 120 | Анилин (80%) + фурфуриловый спирт (20%) | Азотная кислота (98 %) | 3060 |

Для описания движения точки переменной массы воспользуемся уравнением Мещерского:

где – скорость ракеты, – скорость истечения.

– общая масса ракеты, – масса конструкции ракеты, – масса топлива, – масса полезной нагрузки.

Масса ракеты вычисляется по формуле:

Таким образом, в систему ОДУ добавляются уравнения:

где – координаты ракеты.

Начальное положение и скорость ракеты на орбите заданной высоты:

Для решения задачи была использована библиотека Scipy на Python, в которой реализована схема Рунге-Кутты 4 порядка. Так как интервал времени достаточно мал, планета считалась фиксированной точкой. В качестве начального времени берется момент t0, когда спутник сделал 1000 оборотов вокруг планеты, а первое, второе и третье тело находятся на одной прямой. Для поиска оптимального угла был использован перебор всех углов с использованием вычисления в нескольких потоках с помощью библиотеки OpenMP. Наименьшее количество топлива для доставки полезного груза на спутник составило 272.2 кг при угле 66.

Рисунок 5 – График зависимости угла от количества топлива для столкновения ракеты со спутником

Рисунок 6 – Траектория движения ракеты и спутника при оптимальном кол-ве топлива

**Вывод**

В ходе данной лабораторной работы были получены навыки численного расчета траекторий движения небесных тел под действием гравитационных сил. Построены траектории движения планеты, спутника и ракеты, которые получены в результате решения системы дифференциальных уравнений.

Также для построения траектории движения ракеты были произведены расчеты необходимого количества топлива для доставки груза на спутник. В результате минимальное необходимое количество топлива – 272.6 кг при угле 66

**Приложение**

Task1.cpp  
#include <iostream>

#include <cmath>

#include <boost/numeric/odeint.hpp>

#include <fstream>

#include <chrono>

#include <iomanip>

using namespace boost::numeric::odeint;

using state\_type = std::array<double, 8>;

using namespace std;

struct Constants {

static constexpr double G = 6.67e-11; // гравитационная постоянная

static constexpr double M1 = 2.0e30; // масса звезды (кг)

static constexpr double M2 = 6.0e24; // масса планеты (кг)

static constexpr double M3 = 7.3e22; // масса астероида (кг)

static constexpr double R1 = 696340e3; // радиус звезды (м)

static constexpr double R2 = 6378e3; // радиус планеты (м)

static constexpr double R3 = 1737e3; // радиус астероида (м)

static constexpr double R12 = 150e9; // начальное расстояние звезда-планета (м)

static constexpr double R23 = 384e6; // начальное расстояние планета-астероид (м)

static constexpr double U2 = 30e3; // начальная скорость планеты (м/с)

static constexpr double U3 = 1e3; // начальная скорость астероида (м/с)

};

// Вспомогательные функции

class Physics {

public:

static bool intersection;

static bool fast\_intersection;

static double distance(double x1, double y1, double x2, double y2) {

double dx = x2 - x1;

double dy = y2 - y1;

return std::sqrt(dx \* dx + dy \* dy);

}

static void calculateForces(const state\_type &y, state\_type &f, double /\* t \*/) {

// Координаты и скорости планеты и астероида

// Солнце находится в начале координат (0,0)

double x2 = y[0], vx2 = y[1], y2 = y[2], vy2 = y[3]; // планета

double x3 = y[4], vx3 = y[5], y3 = y[6], vy3 = y[7]; // астероид

// Расчёт расстояний

double r\_12 = std::sqrt(x2 \* x2 + y2 \* y2); // расстояние от солнца до планеты

double r\_13 = std::sqrt(x3 \* x3 + y3 \* y3); // расстояние от солнца до астероида

double r\_23 = distance(x2, y2, x3, y3); // расстояние между планетой и астероидом

// Скорости

f[0] = vx2;

f[2] = vy2; // планета

f[4] = vx3;

f[6] = vy3; // астероид

// Ускорения для планеты (тело 2)

f[1] = -Constants::G \* Constants::M1 \* x2 / std::pow(r\_12, 3) +

Constants::G \* Constants::M3 \* (x3 - x2) / std::pow(r\_23, 3);

f[3] = -Constants::G \* Constants::M1 \* y2 / std::pow(r\_12, 3) +

Constants::G \* Constants::M3 \* (y3 - y2) / std::pow(r\_23, 3);

// Ускорения для астероида (тело 3)

f[5] = -Constants::G \* Constants::M1 \* x3 / std::pow(r\_13, 3) +

Constants::G \* Constants::M2 \* (x2 - x3) / std::pow(r\_23, 3);

f[7] = -Constants::G \* Constants::M1 \* y3 / std::pow(r\_13, 3) +

Constants::G \* Constants::M2 \* (y2 - y3) / std::pow(r\_23, 3);

}

// нормализация

static pair<double, double> normalize\_vector(double x, double y) {

double length = sqrt(x \* x + y \* y);

return {x / length, y / length};

}

// проверка на коллианеарность

static vector<double> coordinates\_on\_line(double x2, double y2, double x3, double y3) {

auto planeta\_norm = normalize\_vector(x2, y2);

auto spytnik\_norm = normalize\_vector(x3, y3);

double vector\_product = planeta\_norm.first \* spytnik\_norm.second - planeta\_norm.second \* spytnik\_norm.first;

if (vector\_product < 1e-7 and abs(x3) > abs(x2)) return vector<double>{x2, y2, x3, y3};

else return vector<double>{};

}

static double coordinates\_on\_line\_coef(double x2, double y2, double x3, double y3) {

auto planeta\_norm = normalize\_vector(x2, y2);

auto spytnik\_norm = normalize\_vector(x3, y3);

double planeta\_coef = planeta\_norm.second / spytnik\_norm.first;

double spytnik\_coef = spytnik\_norm.second / spytnik\_norm.first;

if (abs(planeta\_coef - spytnik\_coef) < 1e-5 and abs(x3) > abs(x2)) return abs(planeta\_coef - spytnik\_coef);

}

// проверка на пересечение

static bool intersection\_angle(double x, double y) {

auto norm\_current = normalize\_vector(x, y);

if (norm\_current.first >= 0.99 and !intersection) {

intersection = true;

return true;

} else if (norm\_current.first < 0 and intersection) {

intersection = false;

}

return false;

}

// вычисление нач координат для второй задачи

static bool fast\_crossing\_check(double x, double y) {

auto norm\_current = normalize\_vector(x, y);

if (norm\_current.first >= 0.97)

return true;

return false;

}

};

bool Physics::intersection = false;

int main() {

state\_type y = {

Constants::R1 + Constants::R12 + Constants::R2, 0, 0,

Constants::U2, // планета (x,vx,y,vy)

Constants::R1 + Constants::R12 + 2 \* Constants::R2 + Constants::R23 + Constants::R3, 0, 0,

Constants::U3 + Constants::U2 // астероид (x,vx,y,vy)

};

int count\_cycle, step;

cout << "Введите кол-во пересечений и шаг: ";

cin >> count\_cycle >> step;

double t = 0.0;

double t\_circle\_end = 60. \* 60 \* 24 \* 365;

double t\_end = t\_circle\_end \* count\_cycle;

double h = step;

runge\_kutta\_cash\_karp54<state\_type> stepper;

std::ofstream fout\_main("../labs/lab1/alia/result/task1/path\_full.csv");

fout\_main << "x2 y2 x3 y3\n";

std::vector<std::vector<double> > orbitsX\_Spytnik(4);

std::vector<std::vector<double> > orbitsY\_Spytnik(4);

std::vector<std::vector<double> > orbitsX\_Planeta(4);

std::vector<std::vector<double> > orbitsY\_Planeta(4);

int curr\_i = 0;

int count = 0;

double t\_curr = t;

double point\_collinear;

double y\_min = 1e6;

// double y\_min = 100;

while (t\_curr < t\_end) {

stepper.do\_step(Physics::calculateForces, y, t, h);

if (Physics::intersection\_angle(y[0], y[2])) {

count++;

}

switch (count) {

case 1:

orbitsX\_Spytnik[0].push\_back(y[4]);

orbitsY\_Spytnik[0].push\_back(y[6]);

orbitsX\_Planeta[0].push\_back(y[0]);

orbitsY\_Planeta[0].push\_back(y[2]);

break;

case 100:

orbitsX\_Spytnik[1].push\_back(y[4]);

orbitsY\_Spytnik[1].push\_back(y[6]);

orbitsX\_Planeta[1].push\_back(y[0]);

orbitsY\_Planeta[1].push\_back(y[2]);

break;

case 500:

// h = 1000; // меняем шаг для уточнения нач данных для второй задачиw

orbitsX\_Spytnik[2].push\_back(y[4]);

orbitsY\_Spytnik[2].push\_back(y[6]);

orbitsX\_Planeta[2].push\_back(y[0]);

orbitsY\_Planeta[2].push\_back(y[2]);

break;

case 1000:

orbitsX\_Spytnik[3].push\_back(y[4]);

orbitsY\_Spytnik[3].push\_back(y[6]);

orbitsX\_Planeta[3].push\_back(y[0]);

orbitsY\_Planeta[3].push\_back(y[2]);

break;

}

if (count > 1000) {

point\_collinear = Physics::coordinates\_on\_line\_coef(y[0], y[2], y[4], y[6]);

if (point\_collinear && point\_collinear < y\_min) {

y\_min = point\_collinear;

cout << endl << "Возможное решение на: " << count - 1 << " шаге" << " ,коэффицент: " << fixed <<

setprecision(10) << point\_collinear << endl;

for (auto value: y) {

cout << fixed << setprecision(2) << value << " ";

}

}

}

fout\_main <<

y[0] << " " << y[2] << " " // координаты планеты

<< y[4] << " " << y[6] << "\n"; // координаты астероида

t\_curr += h;

curr\_i++;

}

fout\_main.close();

cout << endl << "Кол-во пересечений: " << count - 1 << endl;

for (int i = 0; i < orbitsX\_Spytnik.size(); i++) {

std::ofstream fout\_spytnik(

"../labs/lab1/alia/result/task1/path\_spytnik\_" + std::to\_string(i) + ".csv");

fout\_spytnik << "x y\n";

for (int j = 0; j < orbitsX\_Spytnik[i].size(); j++) {

fout\_spytnik << orbitsX\_Spytnik[i][j] << " " << orbitsY\_Spytnik[i][j] << std::endl;

}

fout\_spytnik.close();

}

for (int i = 0; i < orbitsX\_Spytnik.size(); i++) {

std::ofstream fout\_planeta(

"../labs/lab1/alia/result/task1/path\_planeta\_" + std::to\_string(i) + ".csv");

fout\_planeta << "x y\n";

for (int j = 0; j < orbitsX\_Planeta[i].size(); j++) {

fout\_planeta << orbitsX\_Planeta[i][j] << " " << orbitsY\_Planeta[i][j] << std::endl;

}

fout\_planeta.close();

}

orbitsX\_Spytnik.clear();

orbitsY\_Spytnik.clear();

return 0;

}

Task2.cpp

#include "task2\_openmp.h"

// #include "task2.h"

int main() {

// double start\_time = omp\_get\_wtime();

//

// analyzeAllAngles(0,360 , 3.0);

//

// double end\_time = omp\_get\_wtime();

// std::cout << "Общее время выполнения: " << (end\_time - start\_time) << " секунд" << std::endl;

// double angle,mt;

// cout << "Введите угол и массу топлива: \n";

// cin >> angle >> mt;

init(287,93,true);

// double mt = findMinFuel(270);

// cout << "Итоговое кол-во топливо: " << mt;

return 0;

}

Task2\_openmp.h  
#pragma once

#include "task2.h"

#include "omp.h"

struct Result {

double angle;

double min\_fuel;

};

double findMinFuel(double angle, double epsilon = 0.05) {

double left = 269;

double right = 290;

double success = 0;

double mt = left;

while (true) {

success = init(mt, angle, false);

if (success == 1 && mt < right) {

// cout << "Оптимальное количество топлива: " << mt << " для угла: " << angle << endl;

break;

}

mt += 0.2;

}

if (mt > right) {

return -1;

}

return mt;

}

void analyzeAllAngles(double start\_angle = 0.0, double end\_angle = 360.0, double angle\_step = 1.0) {

int num\_angles = static\_cast<int>((end\_angle - start\_angle) / angle\_step) + 1;

std::vector<Result> results(num\_angles);

int num\_threads = omp\_get\_max\_threads();

std::cout << "Запуск вычислений на " << num\_threads << " потоках" << std::endl;

#pragma omp parallel for schedule(dynamic) num\_threads(num\_threads)

for (int i = 0; i < num\_angles; i++) {

double angle = start\_angle + i \* angle\_step;

double min\_fuel = findMinFuel(angle);

if (min\_fuel == -1) {

continue; // Пропускаем неудачные расчеты

}

results[i] = {angle, min\_fuel};

#pragma omp critical

{

std::cout << "Поток " << omp\_get\_thread\_num()

<< " | Угол: " << std::fixed << std::setprecision(2) << angle

<< "° | Минимальное топливо: " << min\_fuel << std::endl;

}

}

// Сохраняем результаты в файл

std::ofstream output("../labs/lab1/alia/result/t/min\_fuel\_results.csv");

output << "angle,min\_fuel\n";

for (const auto &result: results) {

output << std::fixed << std::setprecision(2)

<< result.angle << "," << result.min\_fuel << "\n";

}

output.close();

}

Task2.h  
#include <iostream>

#include <cmath>

#include <boost/numeric/odeint.hpp>

#include <fstream>

#include <chrono>

#include <exception>

#include <string>

#include "task2\_const.h"

using namespace boost::numeric::odeint;

using state\_type = std::array<double, 8>;

using namespace std;

// Вспомогательные функции

class Physics {

public:

static double mt;

static double r12x; // координаты планеты

static double r12y;

static double distance(double x1, double y1, double x2, double y2) {

double dx = x2 - x1;

double dy = y2 - y1;

return std::sqrt(dx \* dx + dy \* dy);

}

static double m(double t) {

if (t >= Constants::T) {

return Constants::M0;

} else {

return (Constants::M0 + mt) / (1 - Constants::koef) - (mt \* t) / Constants::T;

}

}

static double dm(double t) {

if (t > Constants::T) {

return 0.0;

} else {

return -mt / Constants::T;

}

}

static void calculateForces(const state\_type &y, state\_type &f, double t) {

// Координаты и скорости спутника и ракеты

double rx = y[0], ry = y[1], vx = y[4], vy = y[5]; // ракета

double r13x = y[2], r13y = y[3], v3x = y[6], v3y = y[7]; // спутник

// Расчёт расстояний

double r = std::sqrt(rx \* rx + ry \* ry); // расстояние от солнца до ракеты

double r2 = distance(rx, ry, r12x, r12y); // расстояние от ракеты до планеты

double r3 = distance(rx, ry, r13x, r13y); // расстояние от ракеты до спутника

double r13 = std::sqrt(r13x \* r13x + r13y \* r13y); // расстояние от спутника до солнца

double r23 = distance(r13x, r13y, r12x, r12y); // расстояние от спутника до планеты

// Расчет скорости ракеты

double v = std::sqrt(vx \* vx + vy \* vy);

// Скорости

f[0] = vx;

f[1] = vy;

f[2] = v3x;

f[3] = v3y;

// Ускорения для ракеты

std::pair<double, double> f\_rocket = {

-(Constants::U \* Physics::dm(t) \* vx) / (v \* Physics::m(t)),

-(Constants::U \* Physics::dm(t) \* vy) / (v \* Physics::m(t))

};

std::pair<double, double> f\_gravity = {

Constants::G \* (

// -Constants::M1 \* rx / std::pow(r, 3)

-Constants::M2 \* (rx - r12x) / std::pow(r2, 3) - Constants::M3 \* (rx - r13x) / std::pow(r3, 3)),

Constants::G \* (

// -Constants::M1 \* ry / std::pow(r, 3)

-Constants::M2 \* (ry - r12y) / std::pow(r2, 3) - Constants::M3 \* (ry - r13y) / std::pow(r3, 3))

};

// Ускорения для ракеты

f[4] = f\_gravity.first + f\_rocket.first;

f[5] = f\_gravity.second + f\_rocket.second;

// Ускорения для спутника

f[6] =

// -Constants::G \* Constants::M1 \* r13x / std::pow(r13, 3)

-Constants::G \* Constants::M2 \* (r13x - r12x) / std::pow(r23, 3); // сила от планеты и солнца

f[7] =

// -Constants::G \* Constants::M1 \* r13y / std::pow(r13, 3)

-Constants::G \* Constants::M2 \* (r13y - r12y) / std::pow(r23, 3); // сила от планеты и солнца

}

};

double Physics::mt;

double Physics::r12x;

double Physics::r12y;

int init(double mt, double angle, bool output = false) {

ofstream fout\_main;

if (output) {

std::ostringstream oss;

oss << std::fixed << std::setprecision(2) << mt; // Ограничиваем 2 знаками после запятой

std::string mt\_str = oss.str();

std::ostringstream oss\_angle;

oss\_angle << std::fixed << std::setprecision(2) << angle;

std::string angle\_str = oss\_angle.str();

fout\_main.open("../labs/lab1/alia/result/task2/full\_trajectory\_mt-" + mt\_str + "\_angle-" + angle\_str + ".csv");

fout\_main << "x y x3 y3\n";

}

double r12x0 = , r12y0 = 141005424242.41,

v2x0 = -26912.62, v2y0 = -11515.37,

r13x0 = -63198811221.89, r13y0 = 141334141912.81,

v3x0 = -27897.52, v3y0 = -11974.71; // нач координаты планеты и спутника

double \_angle = angle \* M\_PI / 180;

Physics::mt = mt;

double v3x = v3x0 - v2x0;

double v3y = v3y0 - v2y0;

// Планета в центре координат

Physics::r12x = r12x0;

Physics::r12y = r12y0;

// Расстояние до спутника

double r3x = r13x0 - Physics::r12x, r3y = r13y0 - Physics::r12y;

double r3 = std::sqrt(r3x \* r3x + r3y \* r3y);

// Рассчитываем начальное положение ракеты

double v0 = std::sqrt(Constants::G \* Constants::M2 / (Constants::R2 + Constants::H));

double rx0 = (Constants::R2 + Constants::H) \* (r3x \* cos(\_angle) - r3y \* sin(\_angle)) / r3;

double ry0 = (Constants::R2 + Constants::H) \* (r3x \* sin(\_angle) + r3y \* cos(\_angle)) / r3;

// Расстояние от ракеты до центра

double r0 = sqrt(rx0 \* rx0 + ry0 \* ry0);

// Начальная скорость ракеты (перпендикулярная радиус-вектору)

double vx0 = -v0 \* ry0 / r0;

double vy0 = v0 \* rx0 / r0;

rx0 += r12x0;

ry0 += r12y0;

double t = 0.0;

double t\_circle\_end = 60. \* 60 \* 24 \* 28 \* 3;

double t\_end = t\_circle\_end;

double h = 7;

state\_type y = {

rx0, ry0, // координаты ракеты

r13x0, r13y0, // координаты спутника

vx0, vy0, // скорость ракеты

v3x, v3y // скорость спутника

};

double t\_curr = t;

runge\_kutta\_cash\_karp54<state\_type> stepper;

int status = 0; //0 - нет столкновения в пределах орбиты спутника,1 - cтолкновение со спутником, 2 - столкновение с планетой

// 3 - ракета улетает далеко

while (t < t\_end) {

stepper.do\_step(Physics::calculateForces, y, t, h);

if (output) {

fout\_main << y[0] << " " << y[1] << " " << y[2] << " " << y[3] << std::endl;

}

// Проверка на столкновение ракеты и планеты

double r\_rocket\_planet = std::sqrt(y[0] \* y[0] + y[1] \* y[1]);

if (r\_rocket\_planet <= Constants::R2) {

status = 2;

break;

}

// Проверка на столкновение ракеты и спутника

double dx\_rocket\_sat = y[0] - y[2];

double dy\_rocket\_sat = y[1] - y[3];

double r\_rocket\_sat = std::sqrt(dx\_rocket\_sat \* dx\_rocket\_sat + dy\_rocket\_sat \* dy\_rocket\_sat);

if (r\_rocket\_sat <= Constants::R3) {

status = 1;

break;

}

// Проверка на то что ракета улетает от планеты далеко

double dx\_dist\_rocket = y[0] - r12x0;

double dy\_dist\_rocket = y[1] - r12y0;

double dist\_rocket = std::sqrt(dx\_dist\_rocket \* dx\_dist\_rocket);

if (dist\_rocket > (Constants::R2 + Constants::R23 + Constants::R3)\*5) {

status = 3;

break;

}

t += h;

}

if (output) {

fout\_main.close();

}

return status;

}

Task2\_const.h  
// Физические константы

struct Constants {

static constexpr double G = 6.67e-11; // гравитационная постоянная

static constexpr double M1 = 2.0e30; // масса звезды (кг)

static constexpr double M2 = 6.0e24; // масса планеты (кг)

static constexpr double M3 = 7.3e22; // масса астероида (кг)

static constexpr double R1 = 696340e3; // радиус звезды (м)

static constexpr double R2 = 6378e3; // радиус планеты (м)

static constexpr double R3 = 1737e3; // радиус астероида (м)

static constexpr double R12 = 150e9; // начальное расстояние звезда-планета (м)

static constexpr double R23 = 384e6; // начальное расстояние планета-астероид (м)

static constexpr double U2 = 30e3; // начальная скорость планеты (м/с)

static constexpr double U3 = 1e3; // начальная скорость астероида (м/с)

static constexpr double T = 2400.0; // время работы двигателя (с)

static constexpr double H = 300e3; // высота орбиты (м)

static constexpr double M0 = 120.0; // масса полезной назрузки (кг)

static constexpr double U = 3060.0; // скорость истечения (м/c)

static constexpr double koef = 0.025;

};