САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕОРИИ СИСТЕМ УПРАВЛЕНИЯ ЭЛЕКТРОФИЗИЧЕСКОЙ АППАРАТУРОЙ

Аминов Руфат Рамильевич

Выпускная квалификационная работа бакалавра

Перелеты в окрестность точки либрации с околоземной орбиты

Направление 010400

Прикладная математика и информатика

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Шмыров В. А.

Санкт-Петербург
2017

Оглавление

Введение

Точки либрации, также называемые точками Лагранжа, являются частными решениями ограниченной задачи трех тел [5]. В системе Земля-Солнце существует 5 точек либрации. Эти точки разделяют на две категории: коллинеарные и треугольные. Важно отметить, что коллинеарные точки являются неустойчивыми, а треугольные — устойчивые. Точка либрации L_1 находится на расстоянии порядка 1.5 миллиона километров от центра Земли на линии Земля-Солнце в сторону Солнца, а L_2, соответственно, находится на расстоянии порядка 1.5 миллиона километров от центра Земли в сторону от Солнца. Окрестности этих точек можно отнести к околоземному пространству, и они представляют интерес для космических проектов. Так, например, ЕКА и НАСА запустили комический аппарат SOHO в точку L_1 для наблюдения за Солнцем, в 2001 году НАСА запустили КА WMAP в окрестность L_2 для наблюдения за реликтовым излучением. В статье [1] рассматривается построение орбиты в окрестности точки либрации L_2 для КА “Миллиметрон”, который предназначен для исследования различных объектов вселенной в миллиметровом и инфракрасном диапазонах. Данный аппарат конструируют в НПО имени С. А. Лавочкина. Окрестности коллинеарных точек либрации также можно использовать для маневров в околоземном космическом пространстве. Существуют задачи, в рамках которых управляемое движение в окрестностях точек либрации используется в контексте противодействия коментно-астероидной опасности. Таким образом, изучения движения в этих областях пространства актуально для современной космической навигации.

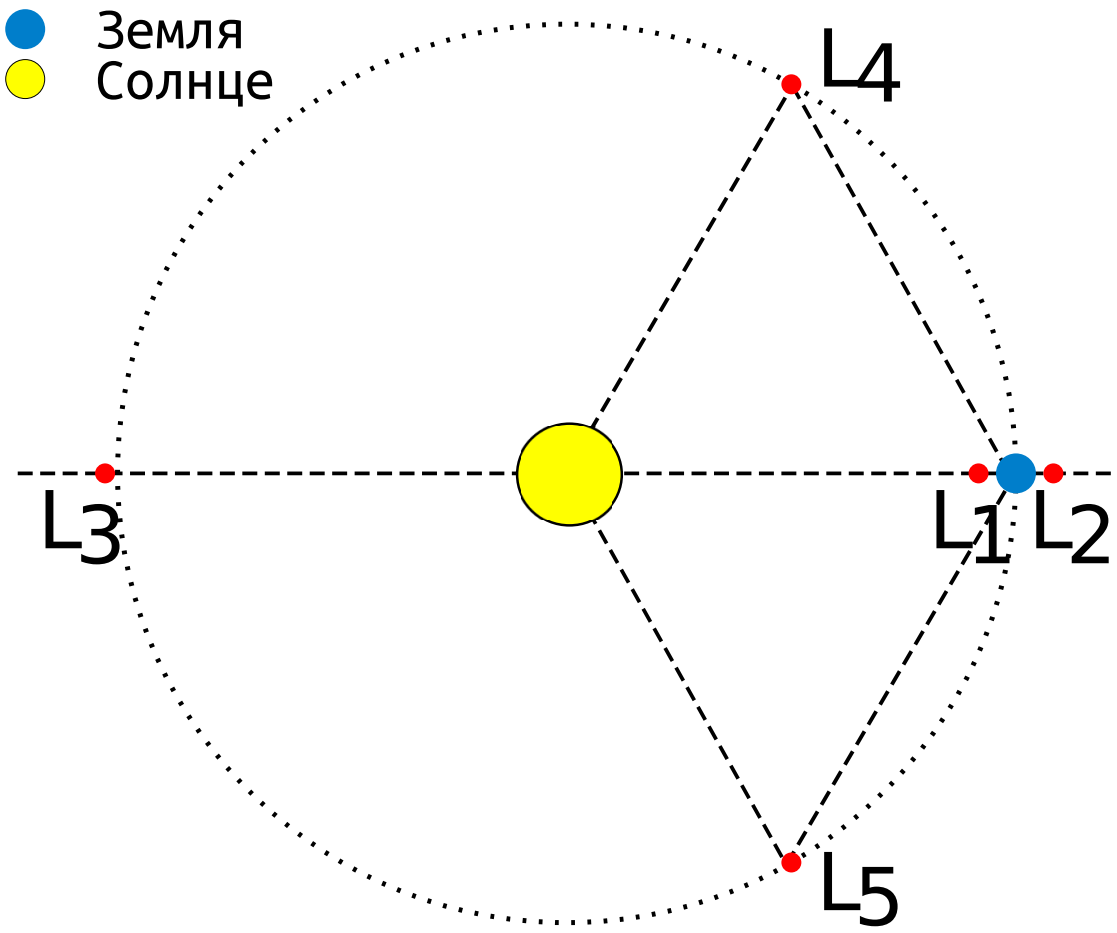

Рис. 1: Расположение точек Лагранжа

Глава 1. Построение траектории перелета с низкой околоземной орбиты

1.1. Постановка задачи

Рассмотрим ограниченную круговую задачу трех тел. Пусть два тела вращаются вокруг их общего барицентра по круговым орбитам под действием ньютоновской силы притяжения. Также введем третье тело, такое, что оно притягивается двумя другими телами, но не оказывает гравитационное влияния на их движение. Ограниченная задача состоит в том, чтобы описать движение третьего тела [5]. В данной работе в качестве математической модели используется один из вариантов специальной аппроксимации ограниченной задачи трех тел во вращающейся геоцентрической системе координат — уравнения Хилла [3].

$$\begin{array}{llll} \dot{x_1} & = y_1 + x_2; & \dot{y_1} & = -\frac{3x_1}{||x||^3} + 2x_1 + y_2; \\ \dot{x_2} & = y_2 - x_1; & \dot{y_2} & = -\frac{3x_2}{||x||^3} - x_2 - y_1; \\ \dot{x_3} & = y_3; & \dot{y_3} & = -\frac{3x_3}{||x||^2} - x_3. \end{array} \tag{1}$$

Где $x = (x_1, x_2, x_3)$ - положение КА в синодической системе координат, $y = (y_1, y_2, y_3)$ - импульсы, $||\cdot||$ - евклидова норма. За единицу расстояния принята величина $1,5 \cdot 10^6$ км, единица времени равняется $\frac{1\text{год}}{2\pi}$, единицей скорости является $303,14$ метров в секунду, единица ускорения равняется $5,93 \cdot 10^{-5} \frac{\text{м}}{\text{с}^2}$. Гамильтониан данной системы (1) имеет вид:

$$H = \frac{1}{2}||y||^2 - \frac{3}{||x||} - \frac{3}{2}x_1^2 + \frac{||x||^2}{2} + x_2 y_1 - x_1 y_2. \tag{2}$$

Координаты L_1 в данной системе координат:

$$x^* = (1, 0, 0), \quad y^* = (0, 1, 0).$$

Линейное приближение уравнений Хилла в окрестности L_1 имеет данный вид:

$$\begin{array}{ll} \dot{x_1} = y_1 + x_2; & \dot{y_1} = -8(x_1 - 1) + (y_2 - 1); \\ \dot{x_2} = y_2 - x_1; & \dot{y_2} = -4x_2 - y_1; \\ \dot{x_3} = y_3; & \dot{y_3} = -4x_3. \end{array} \tag{3}$$

Собственными числами системы (3) являются:

$$\lambda_1 = \sqrt{1+2\sqrt{7}}, \qquad \lambda_2 = -\sqrt{1+2\sqrt{7}}, \qquad \lambda_3 = i\sqrt{2\sqrt{7}-1},$$
$$\lambda_4 = -i\sqrt{2\sqrt{7}-1}, \qquad \lambda_5 = 2i, \qquad \lambda_6 = -2i.$$

Отсюда видно, что движения в плоскости эклиптики ($x_3 = 0$, $y_3 = 0$) является неустойчивым, так как существует собственное число λ_1, вещественная часть которого принадлежит правой полуплоскости. Отметим, что имеет место вырожденность уравнений (3), так как пространственные переменные x_3, y_3 отделяются от переменных x_1, x_2, y_1, y_2, описывающих движение в плоскости эклиптики.

На рисунках 2 и 3 показано, что КА в неуправляемом режиме уходит из малой окрестности точки либрации L_1.

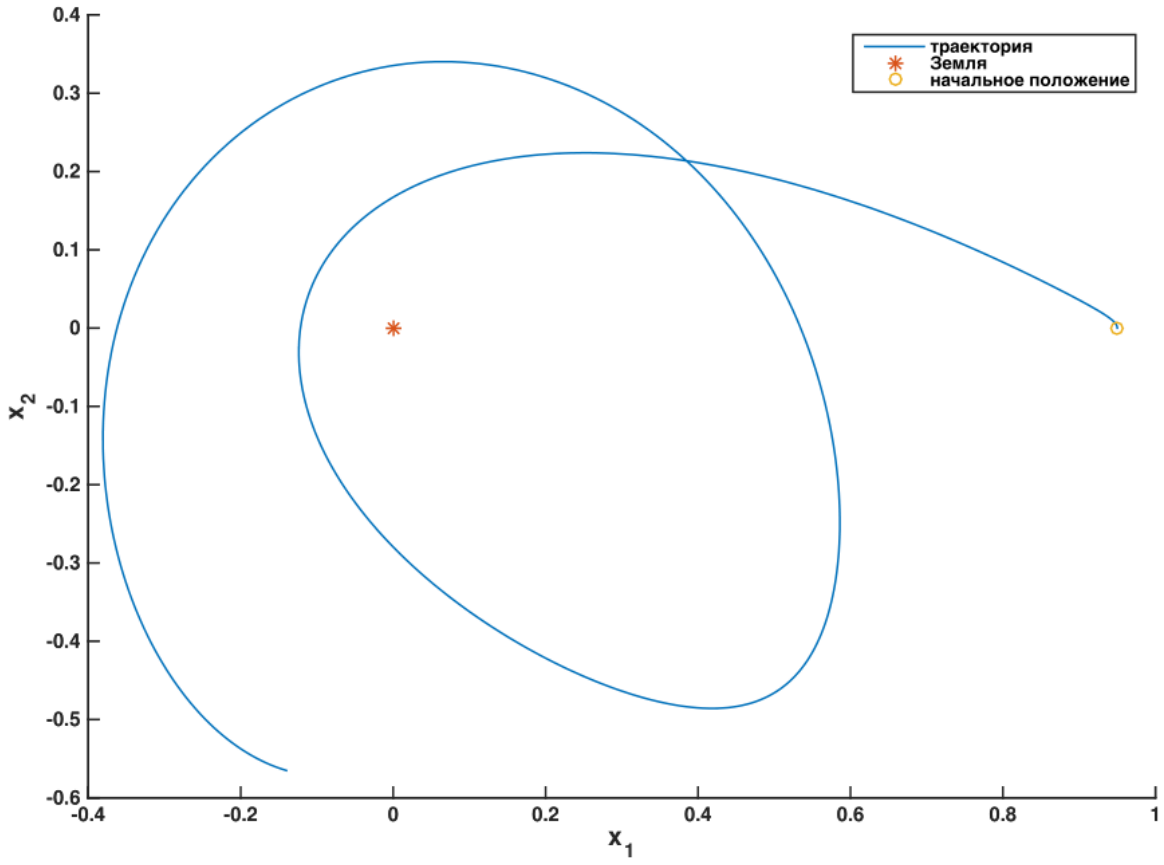

Рис. 2: $x_0 = (0.95, 0.05), y_0 = (0, 1)$

Можно выделить два подхода решения проблемы неустойчивости в окрестности коллинеарной точки либрации L_1. Первым является использование непрерывного управления. Вторым может служить выход на специальную условно-периодическую орбиту, на которой КА будет находится достаточно долго в неуправляемом режиме.

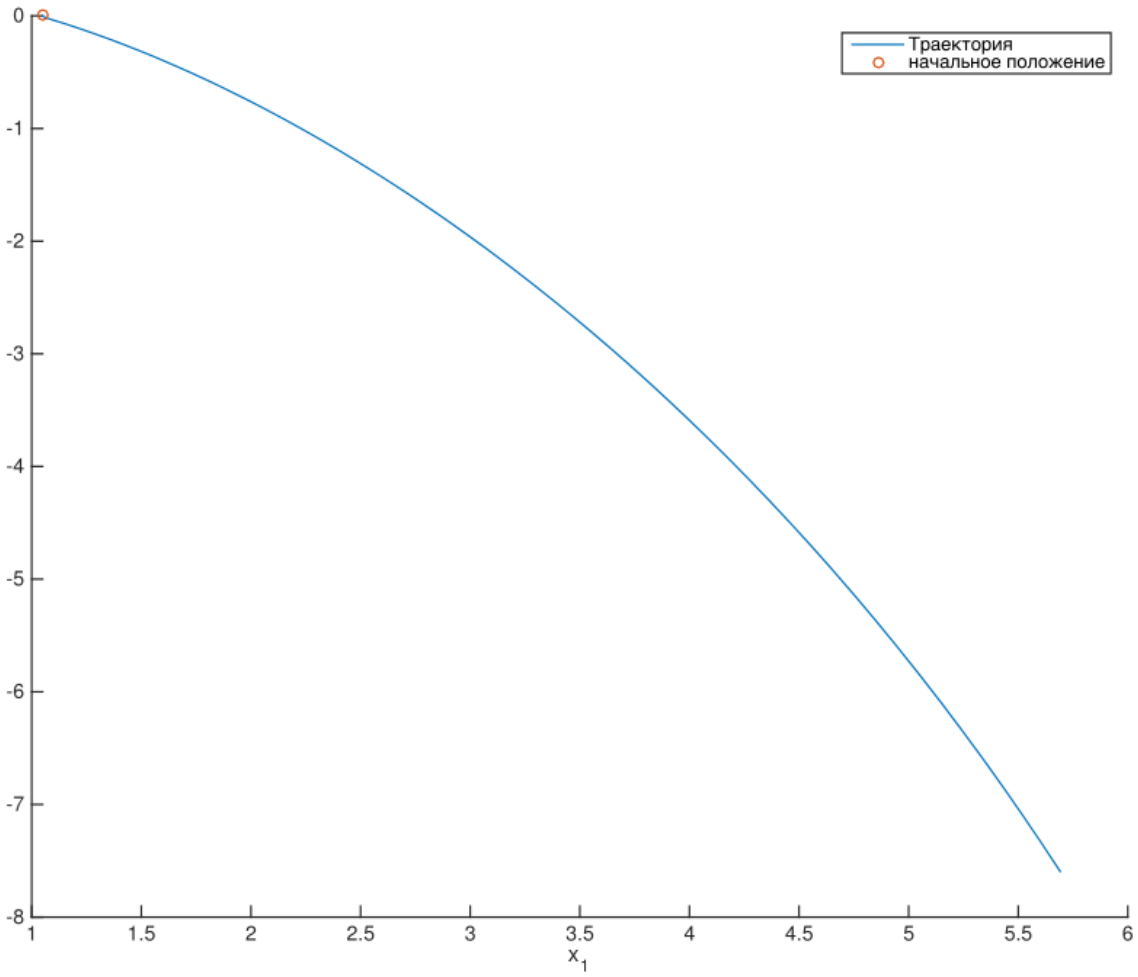

Рис. 3: $x_0 = (1.05, 0.05), y = (0, 1)$

1.2. Построение траектории перелета

В дальнейшем движение будет рассматриваться в плоскости эклиптики ($x_3 \equiv 0, y_3 \equiv 0$). Кривые нулевой скорости (также именуемы линиями Хилла) могут дать приблизительную картину того, в какие области КА никогда не сможет попасть. Для получения уравнений кривых достаточно воспользоваться соотношением $H(x, 0) = C$ [4].

$$H(x,0) = \frac{x_2^2}{2} - x_1^2 - \frac{3}{\sqrt{x_1^2 + x_2^2}} = C.$$

На рисунке 4 видно, что если значение гамильтониана (2)при начальных данных меньше -4 ($H(x_0, y_0) < -4$), то КА не сможет добраться до L_1. Для численного моделирования перелета с низкой околоземной орбиты в окрестность точки либрации L_1 был использован метод Рунге-Кутты четвертого порядка. В результате численного эксперимента была подобрана траектория с начальными данными

$$x_0 = (0.05, 0.045), \quad y_0 = (24.0834, 17.4674).$$

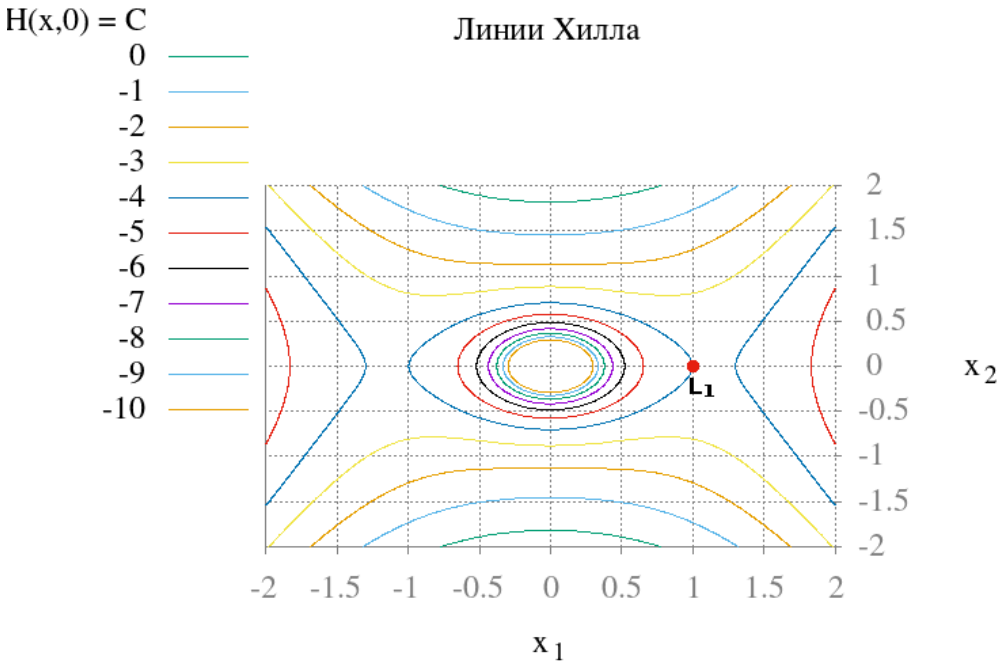

Рис. 4: Линии нулевой скорости для различных значений гамильтониана

по которой КА удалось приблизиться к точке L_1 в пространстве положений (рисунок 5). Следует отметить, что при подлете к окрестности точки либрации в пространстве положений, скорость принимает достаточно большое значение, что приводит к стремительному уходу КА из окрестности L_1. Однако, такие траектории можно использовать в задаче управления КА в окрестности L_1, а также при подлете к окрестности точки либрации. Для проверки корректности численных результатов был использован метод обратной прогонки. Метод обратной прогонки заключается в том, чтобы провести численный эксперимент с момента $t = T$ и начальными состояниями $(x(T), y(T))$ с отрицательным шагом численного интегрирования. Таким образом, в момент $t = 0$ при отсутствии серьезных погрешностей результат интегрирования должен совпасть с начальными состояниями (x_0, y_0). В результате применения метода обратной прогонки корень квадратный от суммы отклонений от начальных данных составил порядка 10^{-9}.

В синодической системе координат гамильтониан (2) явно не зависит от времени, следовательно, на всей траектории он должен оставаться постоянным. Колебания гамильтониана системы на рисунке 6 составили порядка 10^{-12}, что дает качественную картину адекватности проведённого численного моделирования.

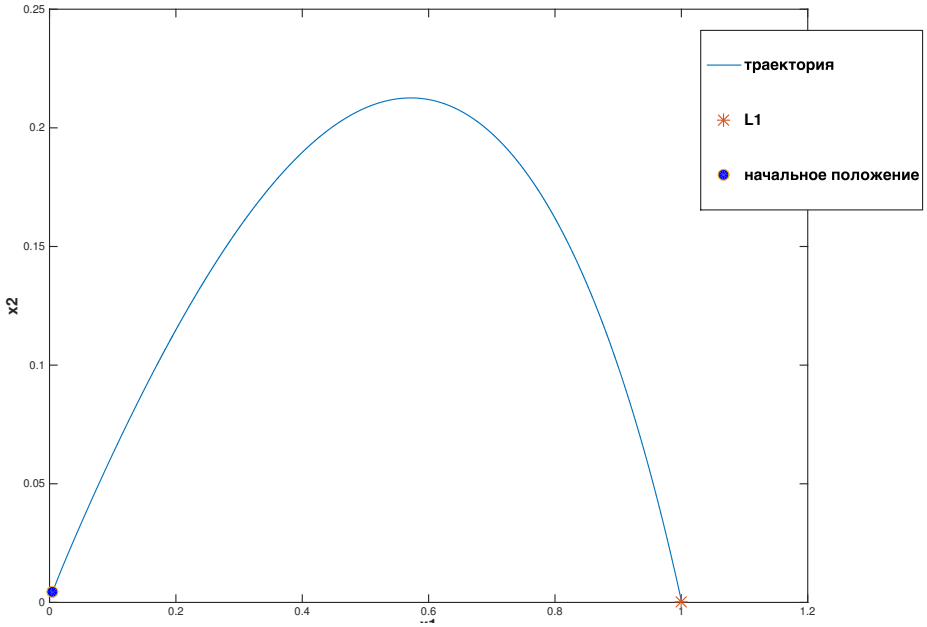

Рис. 5: Траектория движения в пространстве положений

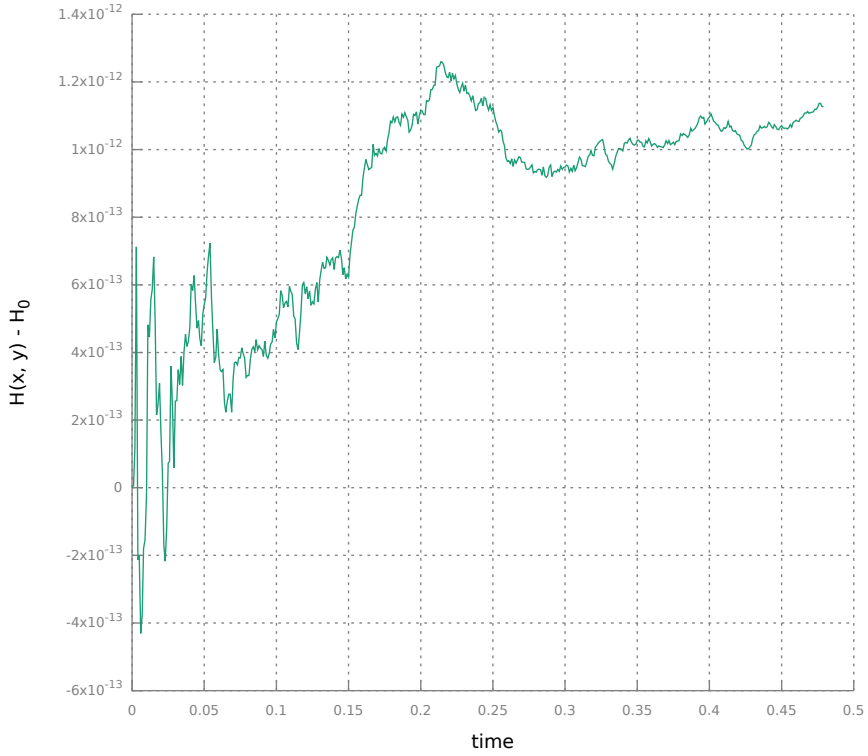

Рис. 6: Колебания гамильтониана системы

Глава 2. Коррекция орбитального движения космического аппарата

2.1. Задача длительного пребывания в окрестности L_1. Уравнения в вариациях

Запишем систему (3) в матричной форме [2].

$$\dot{\xi} = A\xi \tag{4}$$

где

$$A = \begin{pmatrix} K & E \\ Y & K \end{pmatrix}, \xi = \begin{pmatrix} x_1 - 1 \\ x_2 \\ y_1 \\ y_2 - 1 \end{pmatrix}$$

$$K = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, E = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, Y = \begin{pmatrix} 8 & 0 \\ 0 & -4 \end{pmatrix}.$$

Рассмотрим собственный вектор-строку b_1 системы (4) соответствующий собственному числу λ_1.

$$b_1 = \begin{pmatrix} 1; & \frac{\lambda_1^2-3}{\lambda_1(\lambda_1^2+5)}; & \frac{\lambda_1^2+3}{\lambda_1(\lambda_1^2+5)}; & \frac{2}{\lambda_1^2+5} \end{pmatrix}.$$

Определим линейную форму $d = b_1\xi$, которую будем называть "функцией опасности". Величина d удовлетворяет уравнениям

$$\begin{aligned} \dot{d} &= \lambda_1 d, \\ d(t) &= d(t_0)e^{\lambda_1(t-t_0)}. \end{aligned} \tag{5}$$

Из уравнений (5) видно, что функция опасности характеризует экспоненциальный уход из окрестности точки либрации L_1. Таким образом, чтобы оставаться в окрестности точки либрации в рамках линейного приближения требуется $d(t_0) = 0$. Из данного тождества можно найти подходящие изменения начальных состояний, чтобы находится в окрестности L_1 продолжительное время. Для достижения этой цели можно применить аппарат уравнений в вариациях, который представляет собой удобный инструмент для анализа малого возмущения начальных данных на решение

системы. Важно отметить, что КА будет находится в окрестности точки либрации бесконечное время только в линейном приближении. Перепишем систему (1) в виде:

$$\dot{z} = f(z) + u, \tag{6}$$

где

$$z = \begin{pmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{pmatrix}, \quad f(z) = \begin{pmatrix} y_1 + x_2 \\ y_2 - x_1 \\ -\frac{3x_1}{||x||^3} + 2x_1 + y_2 \\ -\frac{3x_2}{||x||^3} - x_2 - y_1 \end{pmatrix}, \quad u = \begin{pmatrix} 0 \\ 0 \\ u_1 \\ u_2 \end{pmatrix}.$$

Пусть для неуправляемой системы (6) имеется решение $z(t) = \phi(z_0, t_0, t)$ с начальным условием $z(t_0) = z_0$. Уравнение в вариациях для системы (6) при $u = 0$ запишем в виде:

$$\dot{\Phi} = \frac{\partial f}{\partial z}(\phi(z_0, t_0, t))\Phi(t) = \begin{pmatrix} K & I \\ Y_x & K \end{pmatrix} \Phi(t), \tag{7}$$

где

$$Y_x = \begin{pmatrix} \frac{9x_1^2}{||x||^5} - \frac{3}{||x||^3} + 2 & \frac{9x_1x_2}{||x||^5} \\ \frac{9x_1x_2}{||x||^5} & \frac{9x_2^2}{||x||^5} - \frac{3}{||x||^3} - 1 \end{pmatrix}, K = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}, I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

с начальным условием $\Phi(t_0) = E$, где $\Phi(t)$ – матрица размерности 4×4, E – единичная матрица размерности 4×4. При изменении начальных данных в конечной точке траектории имеем.

$$\phi(z_0 + \Delta z_0, t_0, T) = \phi(z_0, t_0, T) + \Phi(T)\Delta z_0 + o(\Delta z_0^2). \tag{8}$$

Так как для нахождения на инвариантном многообразии требуется $d(t) = 0$, то из условий (5) и (8) получим.

$$d(T) = b_1(\xi(T) + \Phi(T)\Delta z_0 + o(\Delta z_0^2)) = 0$$

2.2. Алгоритм коррекции орбитального движения КА. Реализация алгоритма.

1. Выберем момент времени t_0, в котором будем изменять состояние системы.

2. Для минимизации функции опасности будем рассматривать $\Delta z_0 = (0, 0, \Delta y_1, \Delta y_2)$

3. Начнем численное интегрирование системы (1) и одновременно интегрирование системы (7) до момента T

4. В итоге, получим линейную комбинацию

$$d(T) = b_1\xi(T) + b_1a_3 \cdot \Delta y_1 + b_1a_4 \cdot \Delta y_2 = 0,$$

где a_i – i-ый столбец матрицы $\Phi(T)$.

5. Далее выбираем какие-либо $\Delta y_1, \Delta y_2$, удовлетворяющие данному тождеству

Для поиска приращений можно воспользоваться каким-либо функционалом. Для первой итерации можно использовать функционал вида:

$$\Delta y_1^2 + (\Delta y_2 - 1)^2 \longrightarrow \min$$

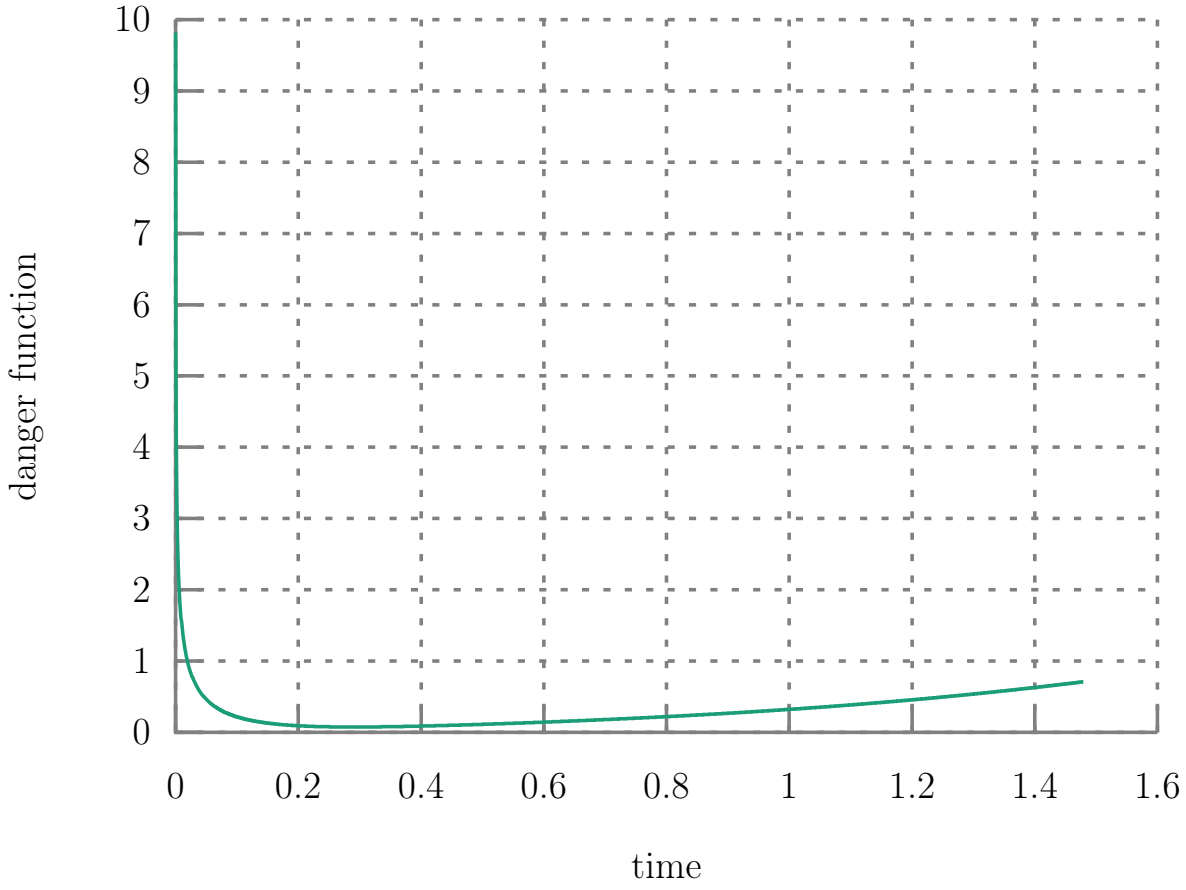

Рис. 7: График функции опасности без использования управления

Выбор момента t_0 представляет особой интерес, ибо пока не удалось получить какого-либо функционала, который мог бы предсказать, когда лучше изменять функцию опасности. Поэтому на начальных этапах момент выбирался эвристически. Один из удачных найденных моментов показан на рисунке 8, на котором удалось оставаться в окрестности точки либрации порядка 3.5 единиц времени. График функции опасности представлен на

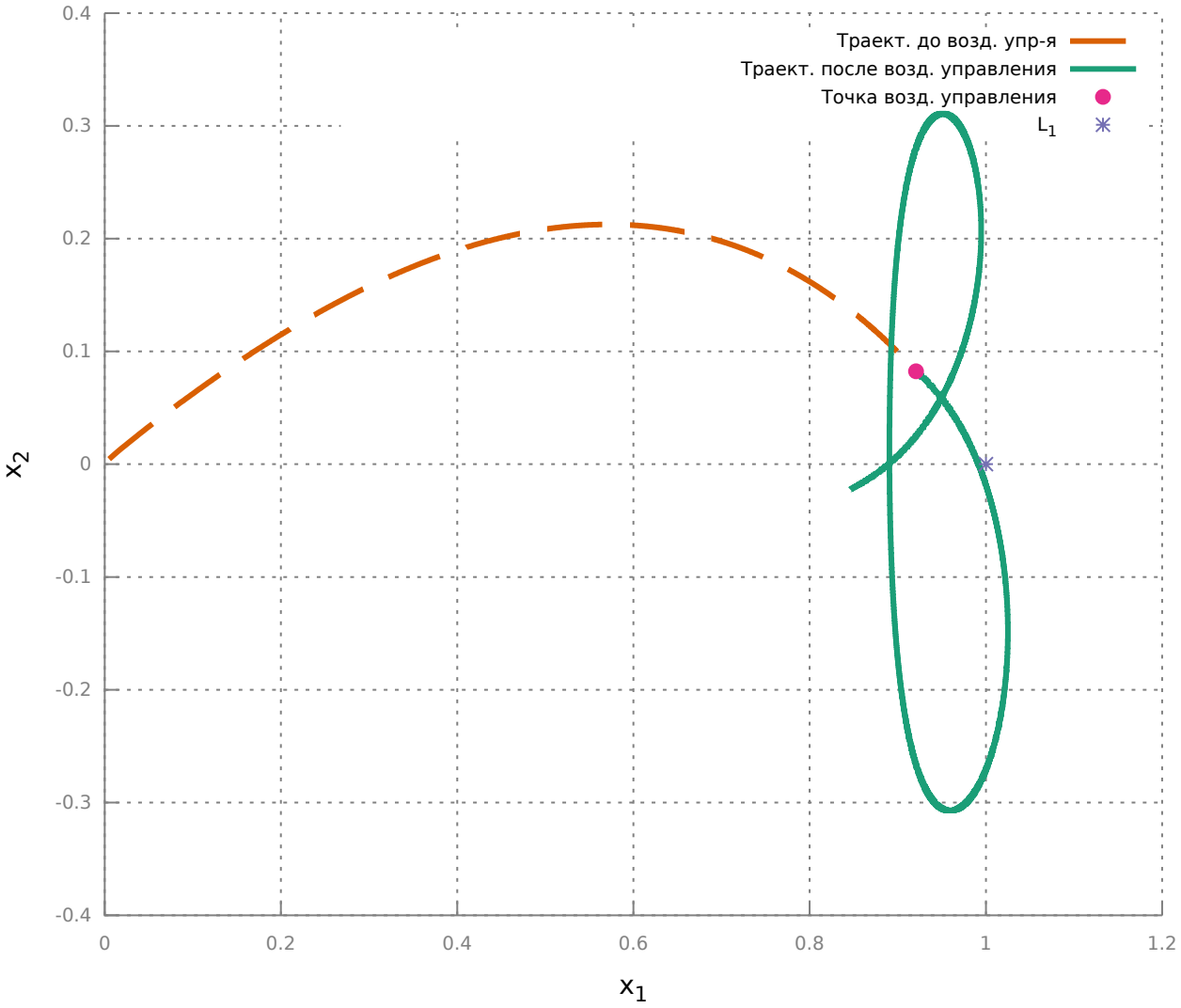

Рис. 8: Траектория космического аппарата с импульсным управлением в момент $t_0 = 0.402531$, $(\Delta y_1, \Delta y_2) = (-0.646173, 0.651189)$

рисунке 9. На нем видно, что после управляющего воздействия функция опасности $|d(t)|$ имеет значение порядка 0.01 три единицы времени.

Для дальнейшей корректировки траектории можно назначить число $\delta > 0$ такое, что при $|d(t)| > \delta$ снова требуется отрабатывать импульсное управление. На рисунке 10 показано исправление траектории при $|d(t)| > 0.3, t > t_0 = 0.402531$ с функционалом

$$\Delta y_1^2 + \Delta y_2^2 \longrightarrow \min .$$

График функции опасности представлен на рисунке 11. На нем видно, что она достаточно долго находится в окрестности 0. Аппарат находился в окрестности L_1 приблизительно 10 единиц времени. Управляющие воздействия представлены в таблице 2.1

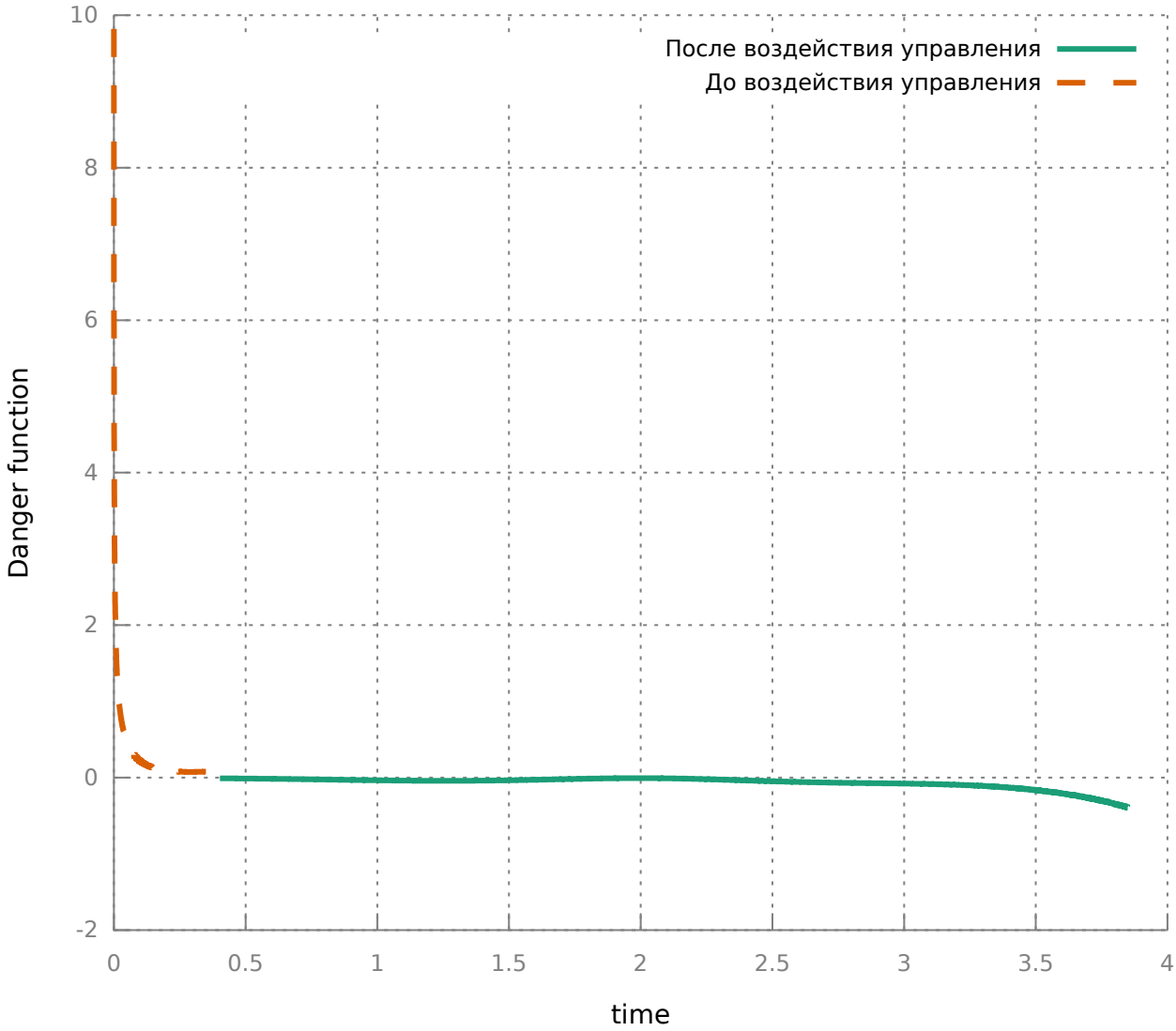

Рис. 9: Функция опасности с отработкой управления в $t_0 = 0.402531$

t	x_1	x_2	y_1	y_2	Δy_1	Δy_2
3.345	0.929411	0.0338744	-0.439277	0.493844	0.709021	0.382807
7.173	0.865134	-0.0172228	-0.459914	0.926901	0.709683	0.383165
8.654	0.887407	0.047298	-0.461464	0.763581	0.708893	0.382738

Таблица 2.1: Моменты времени и состояния системы, в которых происходило управляющее воздействие Δy_1 и Δy_2

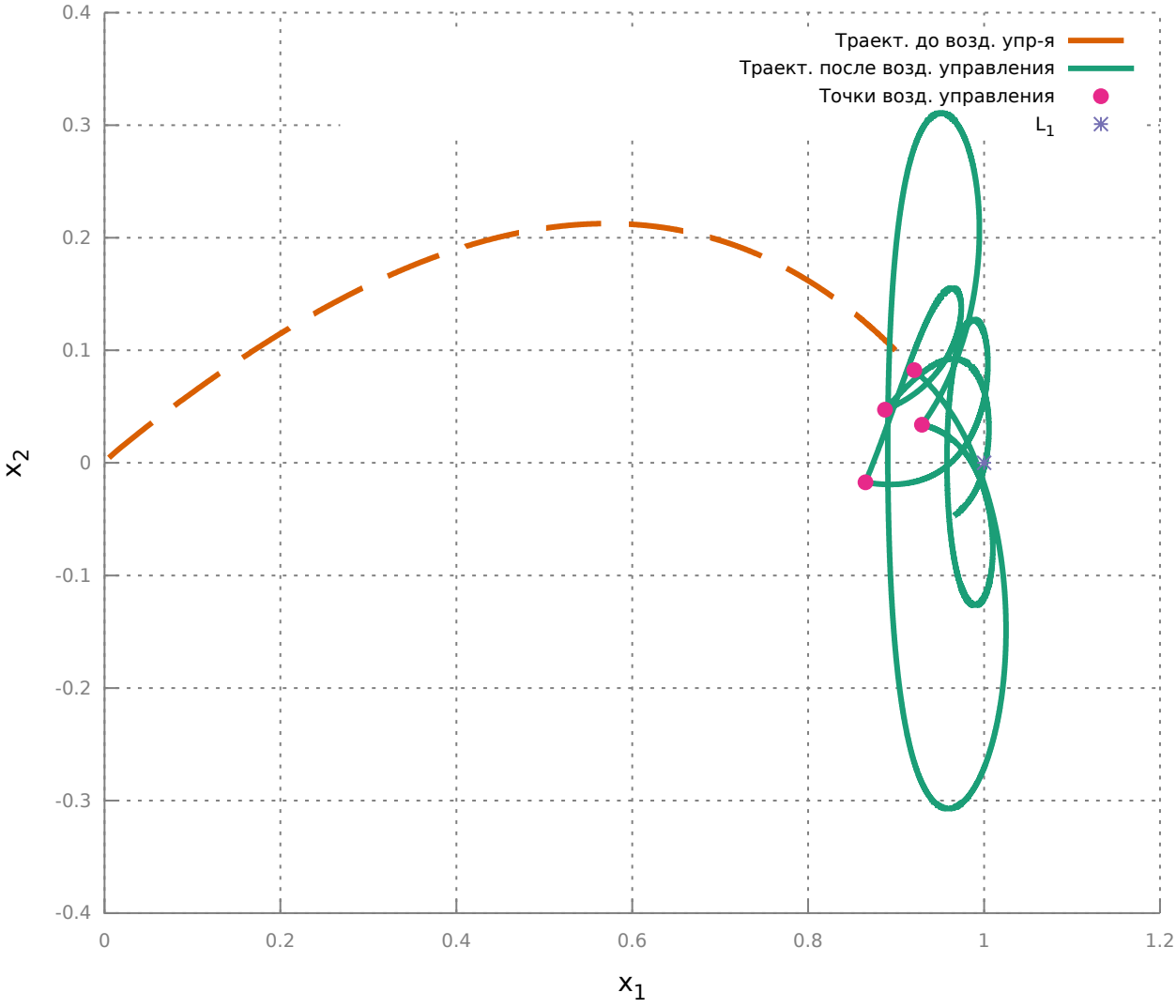

Рис. 10: Изменние траектории при $|d(t)| > 0.3$ после $t_0 = 0.402531$

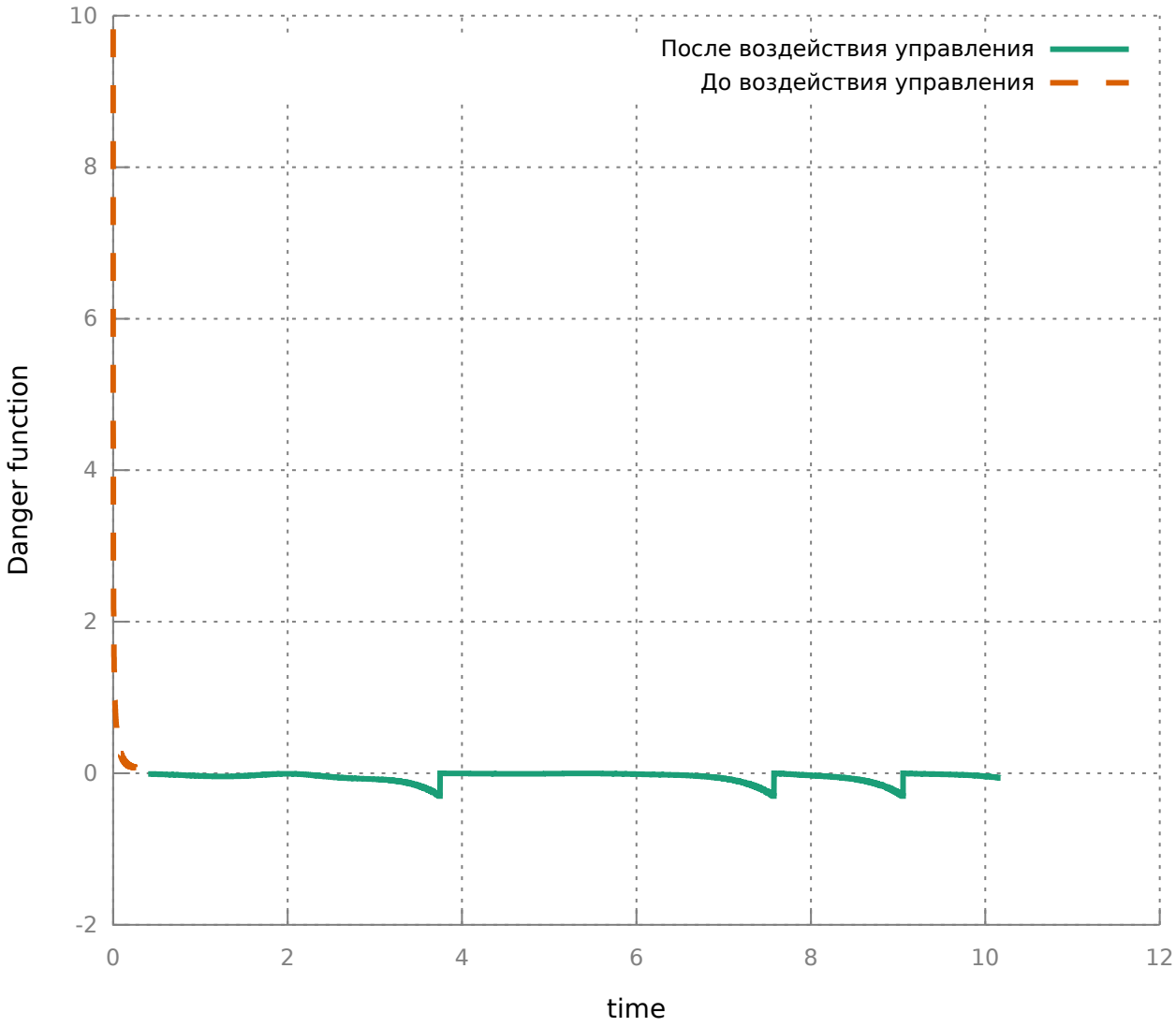

Рис. 11: Изменние траектории при $|d(t)| > 0.3$ после $t_0 = 0.402531$

Заключение

В представленной работе построена траектория перелета с низкой околоземной орбиты в окрестности коллинеарной точки либрации L_1 в пространстве положений. Для достижения инвариантного многообразия, на котором космический аппарат находится в окрестности точки либрации длительное время, были построены импульсные управления в виде мгновенных приращений по скоростям в плоскости эклиптики. Для построения импульсных управлений использовался аппарат уравнений в вариациях. Далее была разработана программа на языке C++, которая в автоматическом режиме при уходе из инвариантного многообразия строит импульсные управления, оптимальные по затратам. С помощью данной программы удалось удержать космический аппарат в окрестности точки либрации L_1 на временном промежутке порядка 1.5 лет при реализации трех импульсных управлений.

Литература

[1] Ильин И.С. Выбор оптимальной орбиты КА "Миллиметрон"из семейства периодических орбит в окрестности точки либрации L_2 системы Солнце – Земля // Препринты ИПМ им. М.В. Келдыша. 2013. No 46. С. 1-21.

[2] Шиманчук Д. В., Шмыров А. С. Построение траектории возвращенияв окрестность коллинеарной точки либрации системы Солнце–Земля // Вестник СПбГУ. Серия 10. Прикладная математика. Информатика. Процессы управления. 2013. №2.

[3] Шмыров В. А. Стабилизация управляемого орбитального движения космического аппарата в окрестности коллинеарной точки либрации L1 // Вестник СПбГУ. Серия 10. Прикладная математика. Информатика. Процессы управления. 2005. №1-2.

[4] Farquhar R. W. The control and use of libration-point satellites: Ph.D. Dissertation. Stanford, CA: Dept. of Aeronautics and Astronautics, Stanford University, 1968. 204 p.

[5] Szebehely V. Theory of orbit – The restricted problem of three bodies. Academic Press, New York, 1967. 342 p.

Приложение

```
/*
Spacecraft's trajectory. Numerical experiment
Copyright (C) 2017  Rufat Aminov

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <iostream>
#include <array>
#include <string>
#include "RK4.hpp"
#include <fstream>
double t0;
int main(int argc, const char *argv[])
{
    VectorXd x(4);
    if (argc == 3)
    {
        std::ifstream fp("/tmp/initialdata");
        fp >> x[0] >> x[1];
        fp >> x[2] >> x[3];
        fp >> t0;
        fp.close();
        std::cout << "x = " << x[0] << " " << x[1] << std::endl
            << "y = " << x[2] << " " << x[3] << std::endl;
        double dy1 = std::atof(argv[1]);
        double dy2 = std::atof(argv[2]);
        x[2] += dy1;
        x[3] += dy2;
        std::cout << "(Delta y1, Delta y2) = (" << dy1 << ", " << dy2 << ")"
   << std::endl;
    }
    else
    {
        x[0] = 0.005;
        x[1] = 0.0045;
        x[2] = 24.0834;
        x[3] = 17.4674;
```

```
        t0 = 0;
    }
    RK4* rk4 = new RK4(x);
    rk4->solve();
    return 0;
}
```

./src/main.cpp

```
#pragma once
#include <array>
#include <iostream>
#include <fstream>
#include <cmath>
#include <limits>
#include "functions.hpp"
#include <Eigen/Dense>

using namespace Eigen;
/**
 * \class Matrix
 * \brief Simple Class for numerical experiment and correction of
 * trajectory if abs(danger function) > 0.3
 */
class RK4 {
public:
    /**
     * \brief constructor. it sets default values
     * \param x initial values
     */
    RK4(const VectorXd& initialValues);
    /**
     * \brief function to start numerical experiment
     */
    void solve();
private:
    // array to store Runge-Kutta 4 states
    std::array<VectorXd, 4> k;
    // integration step
    double stepValue;
    // variable to store amout of iterations
    uint counter;
    // variable to store time
    double time;
    // file to log positions
    std::ofstream mOutX;
    // file to log speeds
    std::ofstream mOutY;
    // file to log danger function
    std::ofstream mDangerOut;
    VectorXd mCurrentValues;

    VectorXd mIntermediateValues;
    /**
     * RK4 step
     * @param  h       integration step
     * @param  current state
     * @return         next state
     */
    VectorXd nextStep(double h, const VectorXd& current);
    /**
```

```
     * save state to the files and correct state if danger function
     * is too big
     * @param newValue value to replace current
     */
    void saveValue(const VectorXd& newValue);
    /**
     * calculates correction of the trajectory to stay near Libration
     * point L1
     */
    void calculateCorrection();
    /**
     * calculates \f[ A \Phi(t) \f]
     * @param  p   positions of the system
     * @param  PHI
     * @return result of the multiplications
     */
    MatrixXd matrix_f(const Vector2f& p, const MatrixXd& PHI);
    MatrixXd m_PHI;
};
```

./src/RK4.hpp

```
#include "RK4.hpp"
#include <iomanip>

const double h = 1e-7;
const double l = std::sqrt(1 + 2 * sqrt(7));
const double coef = l * (l * l + 5);
VectorXd b(4), xi(4);
double previousDangerValue;
extern double t0;

RK4::RK4(const VectorXd& initialValues) {
  previousDangerValue = 0;
  b[0] = 1; b[1] = (l * l - 3) / coef; b[2] = (l*l + 3) / coef; b[3]= 2.0/(l*l
    + 5);
  mCurrentValues = initialValues;

  mIntermediateValues = mCurrentValues;
  stepValue = 1e-7;
  time = 0;

  mDangerOut.open("/tmp/danger.dat");
  mOutX.open("/tmp/x1.dat");
  mOutY.open("/tmp/y1.dat");
}

void RK4::solve() {
  counter = 0;

  while(time < 5) {
    mIntermediateValues = this->nextStep(stepValue, mCurrentValues);
    saveValue(mIntermediateValues);
  }
  mOutX.close();
  mOutY.close();
  mDangerOut.close();
}

VectorXd RK4::nextStep(double h, const VectorXd& current) {
  VectorXd tempValues = current;
```

```
  k[0] = f(tempValues);
  tempValues = current + k[0] * h / (2.0);
  k[1] = f(tempValues);
  tempValues = current + k[1] * h / (2.0);
  k[2] = f(tempValues);
  tempValues = current + k[2] * h;
  k[3] = f(tempValues);

  return current + (k[0] + k[1] * 2.0 + k[2] * 2.0 + k[3]) * (h / 6.0);
}

void RK4::saveValue(const VectorXd& newValue)
{
  mCurrentValues = newValue;
  // recalculate xi
  xi[0] = newValue[0] - 1;
  xi[1] = newValue[1]; xi[2] = newValue[2];
  xi[3] = newValue[3] - 1;
  // calculate danger function
  double dangerValue = b.dot(xi);
  // save values to the file
  if (counter % 10000 == 0)
  {
    mDangerOut << t0 + time << "\t" << dangerValue << std::endl;
    mOutX << newValue[0] << "\t" << newValue[1] << std::endl;
    mOutY << newValue[2] << "\t" << newValue[3] << std::endl;
  }
  // if danger function is huge enough, let's minimize it
  // by impulse control
  if (std::abs(dangerValue) > 0.3) {
    calculateCorrection();
  }
  time += stepValue;
  counter++;
}

void RK4::calculateCorrection()
{
  m_PHI = MatrixXd::Identity(4, 4);
  VectorXd z1 = mCurrentValues;
  VectorXd z2;
  // calculate PHI matrix by Runge-Kutta method of 2nd order
  Vector2f x1(z1[0], z1[1]);
  z2 = this->nextStep(h, z1);
  Vector2f x2(z2[0], z2[1]);
  MatrixXd PHI_1 = matrix_f(x1, m_PHI);
  MatrixXd PHI_2 = matrix_f(x2, m_PHI + h * PHI_1);
  m_PHI += 0.5 * h  * (PHI_1 + PHI_2);
  z1 = z2;
  // last two rows of PHI matrix
  VectorXd a3(4), a4(4);
  for (int i = 0; i < 4; i++) {
    a3(i) = m_PHI(i, 2);
    a4(i) = m_PHI(i, 3);
  }
  // coeff of linear combinations
  double gamma = -b.dot(xi);
  double alpha = b.dot(a3);
  double beta = b.dot(a4);
  std::cout << "\talpha = " << alpha
```

```
    << "\tbeta = " << beta
    << "\tgamma = " << gamma << std::endl;
  gamma /= alpha; beta /= alpha;

  double dy2 = (beta * gamma) / (beta * beta + 1);
  double dy1 = gamma - beta * dy2;
  // values of impulses
  std::cout << "(Delta y1, Delta y2) = ("
    << dy1 << ", " << dy2 << ")" << std::endl;
  // state of the system before impulse
  std::cout << mCurrentValues[0] << "\t" << mCurrentValues[1]
    << std::endl
    << mCurrentValues[2] << "\t" << mCurrentValues[3] << std::endl;
  // let's change impulses
  mCurrentValues[2] += dy1;
  mCurrentValues[3] += dy2;
}

MatrixXd RK4::matrix_f(const Vector2f& p, const MatrixXd& PHI)
{
  double sq = p[0] * p[0] + p[1] * p[1];
  double pow5 = std::pow(sq, -5.0 / 2.0);
  double pow3 = std::pow(sq, -3.0 / 2.0);
  double sq11 = p[0] * p[0];
  double sq12 = p[0] * p[1];
  double sq22 = p[1] * p[1];
  double a20 = 9 * sq11 * pow5 - 3.0 * pow3 + 2;
  double a21 = 9 * sq12 * pow5;
  double a31 = 9 * sq22 * pow5 - 3.0 * pow3 - 1;
  MatrixXd A(4, 4);
  A(0, 0) =  0;  A(0, 1) = 1;   A(0, 2) =  1; A(0, 3) = 0;
  A(1, 0) = -1;  A(1, 1) = 0;   A(1, 2) =  0; A(1, 3) = 1;
  A(2, 0) = a20; A(2, 1) = a21; A(2, 2) =  0; A(2, 3) = 1;
  A(3, 0) = a21; A(3, 1) = a31; A(3, 2) = -1; A(3, 3) = 0;

  return A * PHI;
}
```

./src/RK4.cpp

```
#pragma once
#include <array>
#include <iostream>
#include <Eigen/Dense>

using namespace Eigen;
/**
 * \f[  \dot{x_1}   \f]
 * @param  x state of the system
 * @return   value of \f[  \dot{x_1}   \f]
 */
double x1(const VectorXd& x);
/**
 * \f[  \dot{x_2}   \f]
 * @param  x state of the system
 * @return   value of \f[  \dot{x_2}   \f]
 */
double x2(const VectorXd& x);
/**
 * \f[  \dot{y_1}   \f]
 * @param  x state of the system
```

```
 * @return    value of \f[  \dot{y_1}   \f]
 */
double y1(const VectorXd& x);
/**
 * \f[  \dot{y_2}   \f]
 * @param  x state of the system
 * @return    value of \f[  \dot{y_2}   \f]
 */
double y2(const VectorXd& x);
/**
 * calculates diff eq
 * @param  x state of the system
 * @return    \f[  \dot{x_1},\dot{x_2},\dot{y_1},\dot{y_2}   \f]
 */
VectorXd f(const VectorXd& x);
```

./src/functions.hpp

```
#include "functions.hpp"
#include <cmath>
#include <algorithm>

double x1(const VectorXd& x)
{
  return x[2] + x[1];
}

double x2(const VectorXd& x)
{
  return x[3] - x[0];
}

double y1(const VectorXd& x)
{
  return -3 * x[0] / pow(x[0] * x[0] + x[1] * x[1], 3.0/2.0) + 2 * x[0] + x
    [3];
}

double y2(const VectorXd& x)
{
  return -3 * x[1] / pow(x[0] * x[0] + x[1] * x[1], 3.0/2.0) - x[1] - x[2];
}

VectorXd f(const VectorXd& x)
{
  VectorXd data(4);
  data[0] = x1(x);
  data[1] = x2(x);

  data[2] = y1(x);
  data[3] = y2(x);

  return data;
}
```

./src/functions.cpp