Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра Математического и компьютерного моделирования

МИНИМИЗАЦИЯ ЗАТРАТ ЭНЕРГИИ НА УПРАВЛЕНИЕ УГЛОВЫМ ДВИЖЕНИЕМ СПУТНИКА

Бакалаврская работа

студента 4 курса 413 группы
направление 01.03.02 Прикладная математика и информатика

механико-математического факультета

Исмайылова Гусейна Али оглы

Научный руководитель
Доцент И. А. Панкратов

Зав. кафедрой
зав.каф., д.ф - м.н. Ю. А. Блинков

Саратов 2016

СОДЕРЖАНИЕ

Стр.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

$\mathbb{R}_3$ — трёхмерное вещественное векторное пространство с введённым на нём положительно определённым скалярным произведением.

Норма — функционал, заданный на векторном пространстве и обобщающий понятие длины вектора или абсолютного значения числа.

ИСЗ — искусственный спутник Земли.

Кинематика — раздел механики, изучающий математическое описание (средствами геометрии, алгебры, математического анализа) движения идеализированных тел (материальная точка, абсолютно твердое тело, идеальная жидкость), без рассмотрения причин движения (массы, сил и т. д.).

Базис — множество таких векторов в векторном пространстве, что любой вектор этого пространства может быть единственным образом представлен в виде линейной комбинации векторов из этого множества — базисных векторов.

Инерциальная система отсчёта (ИСО) — система отсчёта, в которой все свободные тела движутся прямолинейно и равномерно, либо покоятся.

ВВЕДЕНИЕ

Выполненная квалификационная работа посвящена задаче оптимального управления углового движения искусственного спутника Земли (ИСЗ), для которой требуется составить и решить краевую задачу с помошью принципа максимума Л.С. Понтрягина. Управление — вектор угловой скорости ИСЗ. Необходимо рассмотреть случай минимизации энергии на перевод ИСЗ в нужное угловое положение. Время окончания процесса фиксировано. Также требуется рассмотреть задачу с её различными параметрами и вывести основные закономерности.

Теория управления является в настоящее время быстро развивающимся разделом современной математики, что вызвано потребностями многочисленных приложений в таких разнообразных дисциплинах как аэрокосмические науки, инженерные и технические науки, гибридные системы, вычислительные и компьютерные науки, океанографические, физические и математические науки. Возрастает интерес к теории оптимального управления и ее приложениям у математиков, экономистов и специалистов по проблемам окружающей среды, а также международных научных организаций, что подтверждается увеличением количества работ в российских и зарубежных издательствах.

Основополагающее значение в теории оптимального управления имеет принцип максимума Л.С. Понтрягина, который получил развитие и приложение в работах российских и зарубежных математиков.

Научая новизна данной работы состоит в рассмотрении поставленной задачи с применением алгебры кватернионов не только в теоретических выкладках, но и в программной реализации решения, которая является универсальной для любого типа задач управления благодаря использованию интерфейсов в программе, которые могут быть ипользованы для самых различных уравнений состояния [19].

В данной работе будут представлены основные определения, понятния и теоремы, которые позволят составить и решить поставленную задачу, решение которой состоит из двух частей: аналитической и численной. Аналитическая часть позволяет перевести задачу оптимального управления к краевой задаче, для которой составлен алгоритм в численной части. Для реализации такого алгоритма была написана программа, которая выдает результаты решения краевой задачи, а также генерирует скрипт для графиков, которые наглядным образом отражают суть этих результатов.

1 Алгебра кватернионов

Будем рассматривать кватернион как математический объект $\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})$, где λ_0 — скалярная величина, $\overline{\lambda}$ — вектор в $\mathbb{R}_3$. Кватернион также можно представить в виде четырехмерного гиперкомплексного числа:

$$\mathbf{\Lambda} = \lambda_0\ +\ i_1\lambda_1\ +\ i_2\lambda_2\ +\ i_3\lambda_3, \tag{1.1}$$

где $\lambda_k,\ k\ =\ \overline{0,3}$ — действительные числа, $i_1,\ i_2,\ i_3$ — кватернионные единицы, обладающие свойством: $i_1^2\ =\ i_2^2\ =\ i_3^2\ =\ i_1 i_2 i_3\ =\ -1.$

Операции с кватернионами:

1) $\mathbf{\Lambda_1}(\lambda_{01},\ \overline{\lambda_1})\ \pm\ \mathbf{\Lambda_2}(\lambda_{02},\ \overline{\lambda_2}) = \mathbf{\Lambda}(\lambda_{01}\ \pm\ \lambda_{02},\ \overline{\lambda_1}\ \pm\ \overline{\lambda_2}),$
2) $\alpha\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})\ =\ \mathbf{\Lambda}(\alpha\lambda_0,\ \alpha\overline{\lambda}),\ \alpha - const,$
3) $\mathbf{\Lambda_1}(\lambda_{01},\ \overline{\lambda_1})\ \circ\ \mathbf{\Lambda_2}(\lambda_{02},\ \overline{\lambda_2})\ =\ \mathbf{\Lambda}(\lambda_{01}\lambda_{02}\ -\ (\overline{\lambda_1},\ \overline{\lambda_2}), \lambda_{01}\overline{\lambda_2}\ +\ \lambda_{02}\overline{\lambda_1}\ +\ \overline{\lambda_1}\ \times\ \overline{\lambda_2})$ (умножение кватернионов),
4) $||\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})||\ =\ \sqrt{\lambda_0^2\ +\ (\overline{\lambda},\ \overline{\lambda})}$ (норма кватерниона),
5) $\widetilde{\mathbf{\Lambda}}\ =\ \mathbf{\Lambda}(\lambda_0,\ -\overline{\lambda})$ (сопряженный кватернион к $\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})$),
6) $\mathbf{\Lambda}^{-1}\ =\ \dfrac{\widetilde{\mathbf{\Lambda}}}{||\mathbf{\Lambda}||^2}$ (обратный кватернион к $\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})$).

Свойства операций с кватернионами:

1) $\mathbf{\Lambda}\ \circ\ (\boldsymbol{M}\ \circ\ \boldsymbol{N})\ =\ (\mathbf{\Lambda}\ \circ\ \boldsymbol{M})\ \circ\ \boldsymbol{N},$
2) $\mathbf{\Lambda}\ \circ\ (\boldsymbol{M}\ +\ \boldsymbol{N})\ =\ \mathbf{\Lambda}\ \circ\ \boldsymbol{M}\ +\ \mathbf{\Lambda}\ \circ\ \boldsymbol{N},$
3) $\mathbf{\Lambda_1}\ \circ\ \mathbf{\Lambda_2}\ \neq\ \mathbf{\Lambda_2}\ \circ\ \mathbf{\Lambda_1}.$

Кватернионы используются для описания поворота твердого тела [1][2]. Именно с помошью алгебры кватернионов будет поставлена задача в работе, это позволит упростить вычисления и программную реализацию её решения.

2 Кинематика вращения твердого тела

Рассмотрим следующие теоремы и утверждения, которые позволят корректно поставить и решить задачу в данной работе [6].

Теорема 1. (*о положении твердого тела*). Произвольное положение твердого тела с неподвижной точкой задается нормированным кватернионом Λ по формулам

$$\overline{e_k} = \mathbf{\Lambda} \circ \overline{i_k} \circ \widetilde{\mathbf{\Lambda}}, \; k = \overline{1,\ 3}, \tag{2.1}$$

где базис e_k связан с самим телом, а базис i_k является неподвижным.

Теорема 2. (*о повороте базиса*). Единичный кватернион вида $\Lambda(\cos(\varphi/2)),\ \overline{e}\sin(\varphi/2))$ задает поворот вокруг единичного вектора $\overline{e}$ на угол φ.

Теорема 3. (*о конечном повороте*). Любое положение твердого тела с неподвижной точкой может быть получено одним поворотом вокруг оси $\overline{e} = \dfrac{\overline{\lambda}}{|\overline{\lambda}|}$ на угол $\varphi = 2\arccos\lambda_0$, где $\mathbf{\Lambda}(\lambda_0,\ \overline{\lambda})$ — нормированный кватернион, задающий положение тела.

Теорема 4. (*о сложении поворотов*). Пусть кватернион $\mathbf{\Lambda_1}$ задает поворот из базиса $i_k^{(1)}$ в базис $i_k^{(2)},\ k = \overline{1,\ 3}$, а кватенион $\mathbf{\Lambda_2}$ — поворот из базиса $i_k^{(2)}$ в $i_k^{(3)}$. Тогда кватернион результирующего поворота $\mathbf{\Lambda}$ будет равен $\mathbf{\Lambda} = \mathbf{\Lambda_1} \circ \mathbf{\Lambda_2}$.

Следствие из теоремы 4. Для того, чтобы выполнить n последовательных поворотов из базиса $i_k^{(1)}$ в базис $i_k^{(n)},\ k = \overline{1,3}$, необходимо найти кватернион Λ, который задается следующей формулой

$$\mathbf{\Lambda} = \mathbf{\Lambda_1} \circ \mathbf{\Lambda_2} \circ \ldots \circ \mathbf{\Lambda_n}, \tag{2.2}$$

где $\mathbf{\Lambda_k},\ k = \overline{1,\ n}$ задает поворот из базиса $I^{(k-1)}$ в базис $I^{(k)}$.

Так как поворот абсолютно твердого тела в трёхмерном евклидовом пространстве задается углами Эйлера [14]: $\alpha,\ \beta,\ \gamma$ (рис. 2.1), то для определения кватерниона, который выражает эти углы, необходимо воспользоватся формулой (2.2), которая в данном случае принимает вид

$$\mathbf{\Lambda} = \mathbf{\Lambda_1} \circ \mathbf{\Lambda_2} \circ \mathbf{\Lambda_3}, \tag{2.3}$$

где по **теореме 2** $\mathbf{\Lambda_1},\ \mathbf{\Lambda_2},\ \mathbf{\Lambda_3}$ определяются следующими соотношениями

$$\begin{cases} \mathbf{\Lambda}_1 = \mathbf{\Lambda_1}\big(cos(\alpha/2),\ (0,\ 0,\ sin(\alpha/2))\big), \\ \mathbf{\Lambda}_2 = \mathbf{\Lambda_2}\big(cos(\beta/2),\ (0,\ sin(\beta/2),\ 0)\big), \\ \mathbf{\Lambda}_3 = \mathbf{\Lambda_3}\big(cos(\gamma/2),\ (sin(\gamma/2),\ 0,\ 0)\big). \end{cases} \tag{2.4}$$

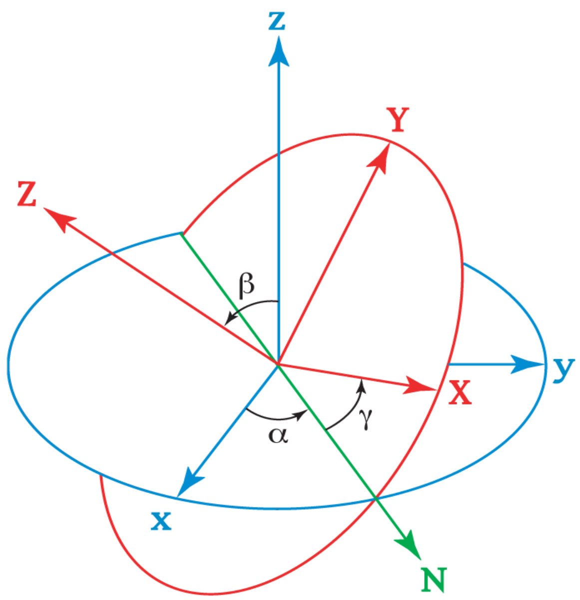


Рисунок 2.1 — углы Эйлера

Из (3.3) получаем связь компонент кватерниона и углов Эйлера [15]

$$\begin{cases} \mathbf{\Lambda} = \mathbf{\Lambda}(\lambda_0,\ \lambda_1,\ \lambda_2,\ \lambda_3), \\ \lambda_0 = cos(\gamma/2)cos(\beta/2)cos(\alpha/2) + sin(\gamma/2)sin(\beta/2)sin(\alpha/2), \\ \lambda_1 = sin(\gamma/2)cos(\beta/2)cos(\alpha/2) - cos(\gamma/2)sin(\beta/2)sin(\alpha/2), \\ \lambda_2 = cos(\gamma/2)sin(\beta/2)cos(\alpha/2) + sin(\gamma/2)cos(\beta/2)sin(\alpha/2), \\ \lambda_3 = cos(\gamma/2)cos(\beta/2)sin(\alpha/2) - sin(\gamma/2)sin(\beta/2)cos(\alpha/2). \end{cases} \tag{2.5}$$

Обратная связь выглядит следующим образом [15]

$$\begin{cases} \gamma \; = \; arctg \; \dfrac{2(\lambda_0\lambda_1 \; + \; \lambda_2\lambda_3)}{1 \; - \; 2(\lambda_1^2 \; + \; \lambda_2^2)}, \\ \\ \beta \; = \; arcsin(2(\lambda_0\lambda_2 \; - \; \lambda_3\lambda_1)), \\ \\ \alpha \; = \; arctg \; \dfrac{2(\lambda_0\lambda_3 \; + \; \lambda_1\lambda_2)}{1 \; - \; 2(\lambda_2^2 \; + \; \lambda_3^2)}. \end{cases} \tag{2.6}$$

Пусть базис $i_k, \; k \; = \; \overline{1, \; 3}$ — неподвижный, относительно которого движется тело с неподвижной точкой O, и для любого момента t задан базис $e_k(t), \; k \; = \; \overline{1, \; 3}$, который связан с телом. Тогда для некоторого промежутка времени Δt тело можно повернуть на некоторый угол $\Delta\varphi(t, \; \Delta t)$, который переведет тело из одного базиса, связанный с телом, в другой при начальном моменте времени t. При этом относительно неподвижного базиса тело будет двигаться с некоторой угловой скоростью $\overline{\omega}$, которая задается следующим образом [7]

$$\overline{\omega} \; = \; \lim_{\Delta t \to 0} \frac{\Delta\varphi(t, \; \Delta t)}{\Delta t} \; \overline{e}(t, \; \Delta t). \tag{2.7}$$

Перемещение из положениия $e_k(t)$ в положение $e_k(t \; + \; \Delta t)$ задается кватернионом

$$\delta\mathbf{\Lambda} \; = \; cos \; \frac{\Delta\varphi(t, \; \Delta t)}{2} \; + \; \overline{e}(t, \; \Delta t) \; sin \; \frac{\Delta\varphi(t, \; \Delta t)}{2}. \tag{2.8}$$

Разложим функцию $cos \; \dfrac{\Delta\varphi(t, \; \Delta t)}{2}$ в ряд Тейлора до второго порядка точности, тогда (2.8) примет вид

$$\delta\mathbf{\Lambda} \; = \; 1 \; + \; \overline{e}(t, \; \Delta t) \; sin \; \frac{\Delta\varphi(t, \; \Delta t)}{2} \; + \; O((\Delta\varphi)^2). \tag{2.9}$$

Так как $\mathbf{\Lambda}(t + \Delta t) \; = \; \delta\mathbf{\Lambda} \circ \Lambda(t)$, то $\mathbf{\Lambda}(t + \Delta t) - \mathbf{\Lambda}(t) \; = \; \delta\mathbf{\Lambda}(t) - \mathbf{\Lambda}(t) \; \Rightarrow$ $\Delta\mathbf{\Lambda} \; = \; (\delta\mathbf{\Lambda} \; - \; 1) \; \circ \; \mathbf{\Lambda}(t)$.

Из (2.9) производная от $\mathbf{\Lambda}$ будет определяться следующим образом

$$\dot{\mathbf{\Lambda}} = \lim_{\Delta t \to 0} \frac{\Delta \mathbf{\Lambda}}{\Delta t} = \lim_{\Delta t \to 0} \frac{\delta \mathbf{\Lambda} - 1}{\Delta t} \circ \mathbf{\Lambda}(t) = \lim_{\Delta t \to 0} \frac{\overline{e}(t,\ \Delta t) sin\ \dfrac{\Delta\varphi(t,\ \Delta t)}{2}}{\Delta t}$$

$$\circ\ \mathbf{\Lambda}(t) = \lim_{\Delta t \to 0} \frac{\overline{e}(t,\ \Delta t)\dfrac{\Delta\varphi(t,\ \Delta t)}{2} sin \dfrac{\Delta\varphi(t,\ \Delta t)}{2}}{\dfrac{\Delta\varphi(t,\ \Delta t)}{2}\Delta t} \circ \mathbf{\Lambda}(t) =$$

$$\lim_{\Delta t \to 0} \frac{\overline{e}(t,\ \Delta t)\Delta\varphi(t,\ \Delta t)}{2\Delta t} \cdot \lim_{\Delta t \to 0} \frac{sin \dfrac{\Delta\varphi(t,\ \Delta t)}{2}}{\dfrac{\Delta\varphi(t,\ \Delta t)}{2}} \circ \mathbf{\Lambda}(t) =$$

$$\frac{1}{2} \lim_{\Delta t \to 0} \frac{\overline{e}(t,\ \Delta t)\Delta\varphi(t,\ \Delta t)}{\Delta t} \circ \mathbf{\Lambda}(t) = \frac{1}{2}\, \overline{\omega} \circ \mathbf{\Lambda}(t).$$

Таким образом, связь между кватернионом Λ и угловой скоростью $\overline{\omega}$ задается формулами

$$\dot{\mathbf{\Lambda}} = \frac{1}{2}\, \overline{\omega} \circ \mathbf{\Lambda}(t), \tag{2.10}$$

$$\overline{\omega} = 2\dot{\mathbf{\Lambda}} \circ \widetilde{\mathbf{\Lambda}}. \tag{2.11}$$

Полученные уравнения (2.10), (2.11) называются уравнениями Пуассона [3][7]. Для удобства выполнения математических операций в этих уравнениях угловую скорость $\overline{\omega}$ можно рассматривать как кватернион, скалярная часть которого равна нулю.

3 Общая задача оптимального управления

Определим задачу оптимального управления в общем случае, которая позволит в дальнейшем корректно сделать постановку задачи, рассматриваемой в данной работе.

Пусть положение тела описывается обыкновенным дифференциальным уравнением [3][4][11]

$$\dot{\mathbf{\Lambda}} = F(t,\ \mathbf{\Lambda}(t),\ \mathbf{\Omega}(t)), \tag{3.1}$$

где $\mathbf{\Lambda}$ — вектор состояния системы, $\mathbf{\Omega}$ — вектор управления, t — время, $t \in T = [t_0,\ t_1]$ — промежуток времени функционирования системы.

$F(t,\ \mathbf{\Lambda},\ \mathbf{\Omega})$ — непрерывная вместе со своими частными производными вектор – функция. Также даны граничные условия

$$\begin{cases} \mathbf{\Lambda}(t_0) = \mathbf{\Lambda_0}, \\ \mathbf{\Lambda}(t_1) = \mathbf{\Lambda_1}. \end{cases} \tag{3.2}$$

Пусть качество управления системы определяется следующим функционалом [11]

$$I = \int_{t_0}^{t_1} F^0(t,\ \mathbf{\Lambda}(t),\ \mathbf{\Omega}(t))dt + F(t_1,\ \mathbf{\Lambda}(t_1)), \tag{3.3}$$

где $F^0(t,\ \mathbf{\Lambda},\ \mathbf{\Omega})dt,\ F(t_1,\ \mathbf{\Lambda})$ — заданные непрерывно дифференцируемые функции, t_1 — фиксировано.

Требуется найти такие $\mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t)$, чтобы функционал качества достигал своего минимума. Искомые функции $\mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t)$ называются соответственно оптимальной траекторией и оптимальным управлением.

Утверждение. [11] Пусть существуют такие $\mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t)$, при которых функционал качества достигает своегоминимального значения, тогда найдется такая вектор — функция $\mathbf{\Psi}(t) = \mathbf{\Psi}(\mathbf{\Psi_1},\ \mathbf{\Psi_2},\ \ldots,\ \mathbf{\Psi_n})$, что:

1) в каждой точке непрерывности управления $\mathbf{\Omega}^*(t)$ функция $H(t,\ \mathbf{\Psi},\ \mathbf{\Lambda}^*,\ \mathbf{\Omega})$ достигает максимума по управлению, то есть

$$\max_{\mathbf{\Omega} \in D(\mathbf{\Omega})} H(t,\ \mathbf{\Psi}(t),\ \mathbf{\Lambda}^*(t),\ \mathbf{\Omega}) = H(t,\ \mathbf{\Psi}(t),\ \mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t)), \tag{3.4}$$

где $D(\mathbf{\Omega})$ — область определения $\mathbf{\Omega}$, а $H(t,\ \mathbf{\Psi},\ \mathbf{\Lambda},\ \mathbf{\Omega})$:

$$H(t,\ \mathbf{\Psi},\ \mathbf{\Lambda},\ \mathbf{\Omega})\ =\ \sum_{j=1}^{n}\boldsymbol{\Psi_j}F_j(t,\ \mathbf{\Lambda},\ \mathbf{\Omega})\ -\ F^0(t,\ \mathbf{\Lambda},\ \mathbf{\Omega}). \tag{3.5}$$

2) функции $\mathbf{\Omega}^*(t),\ \mathbf{\Psi}(t)$ удовлетворяют системе уравнений

$$\begin{cases} \dot{\mathbf{\Lambda}}^*_{\boldsymbol{j}}(t)\ =\ \dfrac{\partial H(t,\ \mathbf{\Psi}(t),\ \mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t))}{\partial\boldsymbol{\Psi_j}}, \\ \\ \dot{\mathbf{\Lambda}}^*_{\boldsymbol{j}}(t)\ =\ F_j(t,\ \mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t)), \\ \\ \mathbf{\Lambda}^*_{\boldsymbol{j}}(t_0)\ =\ \mathbf{\Lambda}_{\mathbf{0}_j}, \\ \\ \dot{\boldsymbol{\Psi}}_{\boldsymbol{j}}(t)\ =\ -\dfrac{\partial H(t,\ \mathbf{\Psi}(t),\ \mathbf{\Lambda}^*(t),\ \mathbf{\Omega}^*(t))}{\partial\boldsymbol{\Lambda_j}},\ j\ =\ \overline{1,\ n}. \end{cases} \tag{3.6}$$

Используемые в формулировке утверждения функции $\boldsymbol{\Psi_1},\ \dots\ ,\ \boldsymbol{\Psi_n}$ называются вспомогательными переменными, $H(t,\ \mathbf{\Psi},\ \mathbf{\Lambda},\ \mathbf{\Omega})$ — гамильтонианом, а (3.6) — системой канонических уравнений [11].

4 Постановка задачи для углового движения тела

Пусть угловое движение тела описывается кинематическим уравнением Пуассона

$$2\dot{\mathbf{\Lambda}} = \mathbf{\Lambda} \circ \mathbf{\Omega}, \tag{4.1}$$

где $\mathbf{\Lambda} = \mathbf{\Lambda}(\lambda_0,\ (\lambda_1,\ \lambda_2,\ \lambda_3))$ — кватернион, характеризующий положение твердого тела относительно инерциальной системы координат, $\mathbf{\Omega} = \mathbf{\Omega}(\omega_0,\ (\omega_1,\ \omega_2,\ \omega_3))$ — кватернион, векторная часть которого равна абсолютной угловой скорости твердого тела относительно этой системы, а скалярная часть равна нулю.

Выражение (4.1) в развернутом виде выглядит следующим образом

$$\begin{cases} 2\dot{\lambda_0} = -\lambda_1\omega_1 - \lambda_2\omega_2 - \lambda_3\omega_3, \\ 2\dot{\lambda_1} = \lambda_0\omega_1 + \lambda_2\omega_3 - \lambda_3\omega_2, \\ 2\dot{\lambda_2} = \lambda_0\omega_2 + \lambda_3\omega_1 - \lambda_1\omega_3, \\ 2\dot{\lambda_3} = \lambda_0\omega_3 + \lambda_1\omega_2 - \lambda_2\omega_1. \end{cases} \tag{4.2}$$

Также дано начальное угловое положение

$$\mathbf{\Lambda}(0) = \mathbf{\Lambda^0}. \tag{4.3}$$

И конечное угловое положение

$$\mathbf{\Lambda}(T) = \mathbf{\Lambda}^{\boldsymbol{T}}. \tag{4.4}$$

Требуется найти такое оптимальное управление $\mathbf{\Omega}(t)$, чтобы функционал качества

$$I = \int\limits_0^T (\alpha_1\omega_1 + \alpha_2\omega_2 + \alpha_3\omega_3)dt \tag{4.5}$$

принимал минимальные значения при фиксированном T, где $\alpha_1,\ \alpha_2,\ \alpha_3 = \mathrm{const} > 0$ — весовые множители функционала (4.5), $\omega_1,\ \omega_2,\ \omega_3$ — компоненты векторной части Ω.

Функционал качества (4.5) характеризует общие энергетические затраты на управление. Для начала задача будет решена в общем случае, а затем будут рассмотрены конкретные примеры.

5 Решение задачи с помощью принципа максимума Понтрягина

Воспользуемся методом максимума Понтрягина, суть которого заключается в том, что задача оптимального управления сводится к решению краевой задачи для системы обыкновенных дифференциальных уравнений. Для этого составляется функция Гамильтона – Понтрягина: [3][4]

$$H = -f_0(\mathbf{\Lambda},\ \mathbf{\Omega},\ t) + \sum_{i=1}^{3} \psi_i f_i(\mathbf{\Lambda},\ \mathbf{\Omega},\ t), \tag{5.1}$$

где $\mathbf{\Lambda} = \mathbf{\Lambda}(\lambda_0,\ (\lambda_1,\ \lambda_2,\ \lambda_3))$, $\mathbf{\Omega} = \mathbf{\Omega}(\omega_0,\ (\omega_1,\ \omega_2,\ \omega_3))$, $\psi_i,\ i = \overline{0,3}$ — вспомогательные сопряженные переменные, которые удовлетворяют следующим условиям

$$\frac{d\psi_i}{dt} = -\frac{\partial H}{\partial \lambda_i}. \tag{5.2}$$

f_0 представляет собой подинтегральное выражения функционала качества (4.5):

$$f_0 = \alpha_1\omega_1 + \alpha_2\omega_2 + \alpha_3\omega_3. \tag{5.3}$$

f_i — это компоненты кватерниона $\frac{1}{2}\mathbf{\Lambda} \circ \mathbf{\Omega}$, то есть

$$\begin{cases} f_0 = -\frac{1}{2}(\lambda_1\omega_1 + \lambda_2\omega_2 + \lambda_3\omega_3), \\ f_1 = -\frac{1}{2}(\lambda_0\omega_1 + \lambda_2\omega_3 - \lambda_3\omega_2), \\ f_2 = -\frac{1}{2}(\lambda_0\omega_2 + \lambda_3\omega_1 - \lambda_1\omega_3), \\ f_3 = -\frac{1}{2}(\lambda_0\omega_3 + \lambda_1\omega_2 - \lambda_2\omega_1). \end{cases} \tag{5.4}$$

Таким образом функция H принимает вид

$$H = \alpha_1\omega_1 + \alpha_2\omega_2 + \alpha_3\omega_3 - \frac{1}{2}\psi_0(\lambda_1\omega_1 + \lambda_2\omega_2 + \lambda_3\omega_3) - \frac{1}{2}\psi_1(\lambda_0\omega_1 + \lambda_2\omega_3 - \lambda_3\omega_2) - \frac{1}{2}\psi_2(\lambda_0\omega_2 + \lambda_3\omega_1 - \lambda_1\omega_3) - \frac{1}{2}\psi_3(\lambda_0\omega_3 + \lambda_1\omega_2 - \lambda_2\omega_1). \quad (5.5)$$

Распишем условия (5.2), для этого найдем частные производные $\dfrac{\partial H}{\partial \lambda_i}, i = \overline{0,\ 3}$:

$$\begin{cases} \dfrac{\partial H}{\partial \lambda_0} = \dfrac{1}{2}\big(\psi_1\omega_1 + \psi_2\omega_2 + \psi_3\omega_3\big), \\ \dfrac{\partial H}{\partial \lambda_1} = \dfrac{1}{2}\big(-\psi_0\omega_1 - \psi_2\omega_3 + \psi_3\omega_2\big), \\ \dfrac{\partial H}{\partial \lambda_2} = \dfrac{1}{2}\big(-\psi_0\omega_2 - \psi_3\omega_1 + \psi_1\omega_3\big), \\ \dfrac{\partial H}{\partial \lambda_3} = \dfrac{1}{2}\big(-\psi_0\omega_3 - \psi_1\omega_2 + \psi_2\omega_1\big). \end{cases} \quad (5.6)$$

Таким образом, уравнения для параметров $\psi_i,\ i = \overline{0,\ 3}$ выглядят следующим способом

$$\begin{cases} 2\dot{\psi}_0 = -\psi_1\omega_1 - \psi_2\omega_2 - \psi_3\omega_3, \\ 2\dot{\psi}_1 = \psi_0\omega_1 + \psi_2\omega_3 - \psi_3\omega_2, \\ 2\dot{\psi}_2 = \psi_0\omega_2 + \psi_3\omega_1 - \psi_1\omega_3, \\ 2\dot{\psi}_3 = \psi_0\omega_3 + \psi_1\omega_2 - \psi_2\omega_1, \end{cases} \quad (5.7)$$

или в кватернионом виде

$$2\dot{\Psi} = \boldsymbol{\Psi} \circ \boldsymbol{\Omega}, \quad (5.8)$$

где $\boldsymbol{\Psi} = \boldsymbol{\Psi}(\psi_0,\ (\psi_1,\ \psi_2,\ \psi_3))$.

Максимальное значение функции H сообщает оптимальное управление[6], поэтому для получения оптимальной угловой скорости найдем для начала стационарные точки функции H.

$$\begin{cases} \dfrac{\partial H}{\partial \omega_1} = \dfrac{1}{2}\big(- \psi_0\lambda_1 + \psi_1\lambda_0 + \psi_2\lambda_3 - \psi_3\lambda_2\big) - 2\alpha_1\omega_1 = 0, \\ \dfrac{\partial H}{\partial \omega_2} = \dfrac{1}{2}\big(- \psi_0\lambda_2 - \psi_1\lambda_3 + \psi_2\lambda_0 + \psi_3\lambda_1\big) - 2\alpha_2\omega_2 = 0, \\ \dfrac{\partial H}{\partial \omega_3} = \dfrac{1}{2}\big(- \psi_0\lambda_3 + \psi_1\lambda_2 - \psi_2\lambda_1 + \psi_3\lambda_0\big) - 2\alpha_3\omega_3 = 0. \end{cases} \tag{5.9}$$

Из (5.9) следует, что у функции H существует единственная стационарная точка $\mathbf{\Omega^0}$, компоненты которой определяются следующим образом

$$\begin{cases} \omega_1^0 = \dfrac{-\psi_0\lambda_1 + \psi_1\lambda_0 + \psi_2\lambda_3 - \psi_3\lambda_2}{4\alpha_1}, \\ \omega_2^0 = \dfrac{-\psi_0\lambda_2 - \psi_1\lambda_3 + \psi_2\lambda_0 + \psi_3\lambda_1}{4\alpha_2}, \\ \omega_3^0 = \dfrac{-\psi_0\lambda_3 + \psi_1\lambda_2 - \psi_2\lambda_1 + \psi_3\lambda_0}{4\alpha_3}. \end{cases} \tag{5.10}$$

Воспользуемся достаточным условием существования максимума [10] для функции H, для этого найдем вторые производные $\dfrac{\partial^2 H}{\partial \omega_i^2}$, $i = \overline{0,\ 3}$ в точке $\mathbf{\Omega_0}$, учитывая условие $\alpha_1,\ \alpha_2,\ \alpha_3 = const > 0$.

$$\begin{cases} \dfrac{\partial^2 H(\mathbf{\Omega^0})}{\partial \omega_1^2} = -2\alpha_1 < 0, \\ \dfrac{\partial^2 H(\mathbf{\Omega^0})}{\partial \omega_2^2} = -2\alpha_2 < 0, \\ \dfrac{\partial^2 H(\mathbf{\Omega^0})}{\partial \omega_3^2} = -2\alpha_3 < 0. \end{cases} \tag{5.11}$$

Из (5.11) следует, что $\mathbf{\Omega_0}$ — точка максимума, таким образом мы приходим к следующей краевой задаче

$$\begin{cases} 2\dot{\mathbf{\Lambda}} \ = \ \mathbf{\Lambda} \ \circ \ \mathbf{\Omega}, \\ 2\dot{\mathbf{\Psi}} \ = \ \mathbf{\Psi} \ \circ \ \mathbf{\Omega}, \\ \mathbf{\Omega} \ = \ \left(0\ , \left(\dfrac{p_1}{4\alpha_1}, \ \dfrac{p_2}{4\alpha_2}, \ \dfrac{p_3}{4\alpha_3}\right)\right), \\ \mathbf{\Lambda}(0) \ = \ \mathbf{\Lambda^0}, \\ \mathbf{\Lambda}(T) \ = \ \mathbf{\Lambda}^{\boldsymbol{T}}, \end{cases} \tag{5.12}$$

где

$$\begin{cases} p_1 \ = \ -\psi_0\lambda_1 \ + \ \psi_1\lambda_0 \ + \ \psi_2\lambda_3 \ - \ \psi_3\lambda_2, \\ p_2 \ = \ -\psi_0\lambda_2 \ - \ \psi_1\lambda_3 \ + \ \psi_2\lambda_0 \ + \ \psi_3\lambda_1, \\ p_3 \ = \ -\psi_0\lambda_3 \ + \ \psi_1\lambda_2 \ - \ \psi_2\lambda_1 \ + \ \psi_3\lambda_0. \end{cases} \tag{5.13}$$

При $\alpha_1 = \alpha_2 = \alpha_3$ краевую задачу (5.12) можно решить аналитически, однако в общем случае получить решение представляется возможным только с помощью численного метода. [6]

6 Алгоритм численного решения задачи

Воспользуемся методом Ньютона [4][17] для решения краевой задачи (5.12). Суть данного итерационного метода состоит в том, что краевая задача сводится к решению серии задач Коши при фиксированном начальном условии с помощью некоторого начального приближения параметра, затем проверяется конечное условие, и если оно удовлетворяется с некоторой требуемой точностью, то задача решена, иначе находится новое приближение, построенное на предыдущем.

Построим сетку на отрезке $[0, T]$ с шагом h и определим на ней функции $\mathbf{\Lambda}, \mathbf{\Psi}, \mathbf{\Omega}$ следующим образом

$$\mathbf{\Lambda}_{\boldsymbol{i}} = \mathbf{\Lambda}(hi),\ \mathbf{\Omega}_{\boldsymbol{i}} = \mathbf{\Omega}(hi),\ \mathbf{\Psi}_{\boldsymbol{i}} = \mathbf{\Psi}(hi),\ i = \overline{0,\ n}, \tag{6.1}$$

где n — количество узлов на сетке.

Опишем первый шаг Ньютона для данной задачи. Зафиксируем начальные условия для функций $\mathbf{\Psi}$, $\mathbf{\Lambda}$, $\mathbf{\Omega}$, причем для функции $\mathbf{\Lambda}$ оно задано выражением (4.3). Таким образом, необходимо найти такое начальное условие для $\mathbf{\Psi}$, чтобы удовлетворялось конечное условие (4.4) для функции $\mathbf{\Lambda}$. Для этого сведем краевую задачу (5.12) к следующей задаче Коши

$$\begin{cases} 2\dot{\mathbf{\Lambda}} = \mathbf{\Lambda} \circ \mathbf{\Omega}, \\ 2\dot{\mathbf{\Psi}} = \mathbf{\Psi} \circ \mathbf{\Omega}, \\ \mathbf{\Lambda}(0) = \mathbf{\Lambda}_{\mathbf{0}}^{(\mathbf{1})}, \\ \mathbf{\Psi}(0) = \mathbf{\Psi}_{\mathbf{0}}^{(\mathbf{1})}, \\ \mathbf{\Omega}(0) = \mathbf{\Omega}_{\mathbf{0}}^{(\mathbf{1})}. \end{cases} \tag{6.2}$$

Для решения задачи (6.2) воспользуемся методом Рунге — Кутты 4–го порядка [5]. Пусть $\mathbf{\Lambda}_{\boldsymbol{i-1}}^{(\mathbf{1})}$, $\mathbf{\Psi}_{\boldsymbol{i-1}}^{(\mathbf{1})}$, $\mathbf{\Omega}_{\boldsymbol{i-1}}^{(\mathbf{1})}$, $i = \overline{1, n}$ — известны, тогда i – ый шаг метода Рунге — Кутты представляет собой следующую последовательность действий:

1) Находим $\boldsymbol{\Lambda}_i^{(1)}$ по следующей схеме

$$\begin{cases} \boldsymbol{\Lambda}_i^{(1)} = \boldsymbol{\Lambda}_{i-1}^{(1)} + \dfrac{h}{6}\Big(\boldsymbol{K}_{\boldsymbol{\Lambda}1} + 2\boldsymbol{K}_{\boldsymbol{\Lambda}2} + 2\boldsymbol{K}_{\boldsymbol{\Lambda}3} + \boldsymbol{K}_{\boldsymbol{\Lambda}4}\Big), \\ \boldsymbol{K}_{\boldsymbol{\Lambda}1} = \dfrac{1}{2}\boldsymbol{\Lambda}_{(i-1)}^{(1)} \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Lambda}2} = \dfrac{1}{2}\Big(\boldsymbol{\Lambda}_{i-1}^{(1)} + \dfrac{h}{2}\boldsymbol{K}_{\boldsymbol{\Lambda}1}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Lambda}3} = \dfrac{1}{2}\Big(\boldsymbol{\Lambda}_{i-1}^{(1)} + \dfrac{h}{2}\boldsymbol{K}_{\boldsymbol{\Lambda}2}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Lambda}4} = \dfrac{1}{2}\Big(\boldsymbol{\Lambda}_{i-1}^{(1)} + h\boldsymbol{K}_{\boldsymbol{\Lambda}2}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}. \end{cases} \tag{6.3}$$

2) Аналогичным способом находим $\boldsymbol{\Psi}_i^{(1)}$

$$\begin{cases} \boldsymbol{\Psi}_i^{(1)} = \boldsymbol{\Psi}_{i-1}^{(1)} + \dfrac{h}{6}\Big(\boldsymbol{K}_{\boldsymbol{\Psi}1} + 2\boldsymbol{K}_{\boldsymbol{\Psi}2} + 2\boldsymbol{K}_{\boldsymbol{\Psi}3} + \boldsymbol{K}_{\boldsymbol{\Psi}4}\Big), \\ \boldsymbol{K}_{\boldsymbol{\Psi}1} = \dfrac{1}{2}\boldsymbol{\Psi}_{(i-1)}^{(1)} \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Psi}2} = \dfrac{1}{2}\Big(\boldsymbol{\Psi}_{i-1}^{(1)} + \dfrac{h}{2}\boldsymbol{K}_{\boldsymbol{\Psi}1}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Psi}3} = \dfrac{1}{2}\Big(\boldsymbol{\Psi}_{i-1}^{(1)} + \dfrac{h}{2}\boldsymbol{K}_{\boldsymbol{\Psi}2}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}, \\ \boldsymbol{K}_{\boldsymbol{\Psi}4} = \dfrac{1}{2}\Big(\boldsymbol{\Psi}_{i-1}^{(1)} + h\boldsymbol{K}_{\boldsymbol{\Psi}2}\Big) \circ \boldsymbol{\Omega}_{i-1}^{(1)}. \end{cases} \tag{6.4}$$

3) На основе $\boldsymbol{\Lambda_i^{(1)}}$, $\boldsymbol{\Psi_i^{(1)}}$ находим $\boldsymbol{\Omega_i^{(1)}}$, компоненты которой определяются формулами (5.10), то есть

$$
\begin{cases}
\boldsymbol{\Omega_i^{(1)}} \; = \; \boldsymbol{\Omega_i^{(1)}}(\omega_{i_1}^{(1)},\ \omega_{i_2}^{(1)},\ \omega_{i_3}^{(1)}),\\[2ex]
\boldsymbol{\Lambda_i^{(1)}} \; = \; \boldsymbol{\Lambda_i^{(1)}}(\lambda_{i_0}^{(1)},\ (\lambda_{i_1}^{(1)},\ \lambda_{i_2}^{(1)},\ \lambda_{i_3}^{(1)})),\\[2ex]
\boldsymbol{\Psi_i^{(1)}} \; = \; \boldsymbol{\Psi_i^{(1)}}(\psi_{i_0}^{(1)},\ (\psi_{i_1}^{(1)},\ \psi_{i_2}^{(1)},\ \psi_{i_3}^{(1)})),\\[2ex]
\omega_{i_1}^{(1)} \; = \; \dfrac{-\psi_{i_0}^{(1)}\lambda_{i_1}^{(1)} \; + \; \psi_{i_1}^{(1)}\lambda_{i_0}^{(1)} \; + \; \psi_{i_2}^{(1)}\lambda_{i_3}^{(1)} \; - \; \psi_{i_3}^{(1)}\lambda_{i_2}^{(1)}}{4\alpha_1},\\[2ex]
\omega_{i_2}^{(1)} \; = \; \dfrac{-\psi_{i_0}^{(1)}\lambda_{i_2}^{(1)} \; - \; \psi_{i_1}^{(1)}\lambda_{i_3}^{(1)} \; + \; \psi_{i_2}^{(1)}\lambda_{i_0}^{(1)} \; + \; \psi_{i_3}^{(1)}\lambda_{i_1}^{(1)}}{4\alpha_2},\\[2ex]
\omega_{i_3}^{(1)} \; = \; \dfrac{-\psi_{i_0}^{(1)}\lambda_{i_3}^{(1)} \; + \; \psi_{i_1}^{(1)}\lambda_{i_2}^{(1)} \; - \; \psi_{i_2}^{(1)}\lambda_{i_1}^{(1)} \; + \; \psi_{i_3}^{(1)}\lambda_{i_0}^{(1)}}{4\alpha_3}.
\end{cases}
\tag{6.5}
$$

Пусть $\boldsymbol{\Lambda}^{(1)}$, $\boldsymbol{\Psi}^{(1)}$, $\boldsymbol{\Omega}^{(1)}$ — полученные в ходе метода Рунге — Кутты функции, заданные на сетке, определяемой выражением (6.1). Тогда главная невязка на первом шаге метода Ньютона $\boldsymbol{D^{(1)}}$ сеточной функции $\boldsymbol{\Lambda^{(1)}}$ будет определятся следующим образом

$$
\begin{cases}
\boldsymbol{D^{(1)}} \; = \; \boldsymbol{D^{(1)}}(d_0^{(1)},\ (d_1^{(1)},\ d_2^{(1)},\ d_3^{(1)})),\\
\boldsymbol{\Lambda^{(1)}}(T) \; = \; \boldsymbol{\Lambda^{(1)}}(\lambda_{n_0}^{(1)},\ (\lambda_{n_1}^{(1)},\ \lambda_{n_2}^{(1)},\ \lambda_{n_3}^{(1)})),\\
\boldsymbol{\Lambda^{T}} \; = \; \boldsymbol{\Lambda^{T}}(\lambda_0^T,(\lambda_1^T,\lambda_2^T,\lambda_3^T)),\\
d_0^{(1)} \; = \; \lambda_{n_0}^{(1)} \; - \; \lambda_0^T,\\
d_1^{(1)} \; = \; \lambda_{n_1}^{(1)} \; - \; \lambda_1^T,\\
d_2^{(1)} \; = \; \lambda_{n_2}^{(1)} \; - \; \lambda_2^T,\\
d_3^{(1)} \; = \; \lambda_{n_3}^{(1)} \; - \; \lambda_3^T.
\end{cases}
\tag{6.6}
$$

Обозначим через $F(\mathbf{\Psi})$ функцию, которая возвращает невязку $\boldsymbol{D}$ по формулам (6.6) и принимает в качестве параметра некоторое приближение начального условия функции $\mathbf{\Psi}$. Таким образом,

$$F(\mathbf{\Psi_0^{(1)}}) \;=\; \boldsymbol{D}^{(1)}. \tag{6.7}$$

Пусть

$$\begin{cases} \mathbf{\Psi_0^{(1)}} \;=\; \mathbf{\Psi_0^{(1)}}(\psi_0^{(1)},\ (\psi_1^{(1)},\ \psi_2^{(1)},\ \psi_3^{(1)})), \\ \mathbf{\Psi_{00}^{(1)}} \;=\; \mathbf{\Psi_{00}^{(1)}}(\psi_0^{(1)} \;+\; \delta,\ (\psi_1^{(1)},\ \psi_2^{(1)},\ \psi_3^{(1)})), \\ \mathbf{\Psi_{01}^{(1)}} \;=\; \mathbf{\Psi_{01}^{(1)}}(\psi_0^{(1)},\ (\psi_1^{(1)} \;+\; \delta,\ \psi_2^{(1)},\ \psi_3^{(1)})), \\ \mathbf{\Psi_{02}^{(1)}} \;=\; \mathbf{\Psi_{02}^{(1)}}(\psi_0^{(1)},\ (\psi_1^{(1)},\ \psi_2^{(1)} \;+\; \delta, \psi_3^{(1)})), \\ \mathbf{\Psi_{03}^{(1)}} \;=\; \mathbf{\Psi_{03}^{(1)}}(\psi_0^{(1)},\ (\psi_1^{(1)},\ \psi_2^{(1)},\ \psi_3^{(1)} \;+\; \delta)), \end{cases} \tag{6.8}$$

где δ — малая величина ($\delta \;\ll\; 1$).

Пусть $\boldsymbol{D_i^{(1)}}$ — невязка некоторого соответствующего ей решения $\boldsymbol{\Lambda_i^{(1)}}$ задачи (6.2) по условию (4.4), полученная при соответствующем начальном условии $\mathbf{\Psi_{0i}^{(1)}}$, $i \;=\; \overline{0,3}$ для функции $\mathbf{\Psi}$ на первом шаге метода Ньютона ($\mathbf{\Psi_0^1}$ из (6.2)), то есть

$$F(\mathbf{\Psi_{0i}^{(1)}}) \;=\; \boldsymbol{D_i^{(1)}}. \tag{6.9}$$

Производные невязки $\dfrac{dF_i^{(1)}}{d\mathbf{\Psi}}$, $i \;=\; \overline{0,\ 3}$ от главной невязки (6.7) определяются следующим образом

$$\frac{dF_i^{(1)}}{d\mathbf{\Psi}} \;=\; \frac{\boldsymbol{D_i^{(1)}} \;-\; \boldsymbol{D}^{(1)}}{\delta}. \tag{6.10}$$

Если выполняется условие

$$\sum_{i=0}^{3} \left|d_i^{(1)}\right| \;<\; \varepsilon, \tag{6.11}$$

где ε — требуемая точность решения, то весь процесс завершается. В противном случае находится следующее приближение начального условия функции $\mathbf{\Psi}$ в виде

$$\mathbf{\Psi_0^{(2)}} \;=\; \mathbf{\Psi_0^{(1)}} \;+\; \chi\mathbf{\Gamma}, \tag{6.12}$$

где множитель χ выбирается таким образом, чтобы невязка $F(\mathbf{\Psi_0^{(2)}}) \;=\; \boldsymbol{D}^{\mathbf{(2)}} \;=\; D^{(2)}(d_0^{(2)},\; (d_1^{(2)},\; d_2^{(2)},\; d_3^{(2)}))$ удовлетворяло следующему неравенству

$$\sum_{i=0}^{3}\left|d_i^{(2)}\right| \;<\; \sum_{i=0}^{3}\left|d_i^{(1)}\right|, \tag{6.13}$$

$\mathbf{\Gamma} \;=\; \mathbf{\Gamma}(\gamma_0,\; (\gamma_1,\; \gamma_2,\; \gamma_3))$ — кватернион, который является решением следующего уравнения

$$F(\mathbf{\Psi_0^{(1)}} \;+\; \mathbf{\Gamma}) \;=\; 0. \tag{6.14}$$

Разложим левую часть (6.14) по формуле Тейлора [10] в точке $\mathbf{\Psi_0^{(1)}}$, полагая, что компоненты кватерниона $\mathbf{\Gamma}$ являются малыми величинами.

$$F(\mathbf{\Psi_0^{(1)}}) \;+\; \sum_{i=0}^{3}\frac{dF_i^{(1)}}{d\mathbf{\Psi}}\gamma_i \;+\; R_2(\mathbf{\Psi_0^{(1)}} \;+\; \mathbf{\Gamma}) = 0. \tag{6.15}$$

В результате получаем систему линейных алгебраических уравнений

$$\frac{dF_0^{(1)}}{d\mathbf{\Psi}}\gamma_0 \;+\; \frac{dF_1^{(1)}}{d\mathbf{\Psi}}\gamma_1 \;+\; \frac{dF_2^{(1)}}{d\mathbf{\Psi}}\gamma_2 \;+\; \frac{dF_3^{(1)}}{d\mathbf{\Psi}}\gamma_3 \;=\; -F(\mathbf{\Psi_0^{(1)}}). \tag{6.16}$$

Таким образом, компоненты кватерниона $\mathbf{\Gamma}$ находятся из системы (6.16).

Метод Ньютона для задачи (5.12) можно представить в виде блок–схемы [22] на рисунке 6.1.

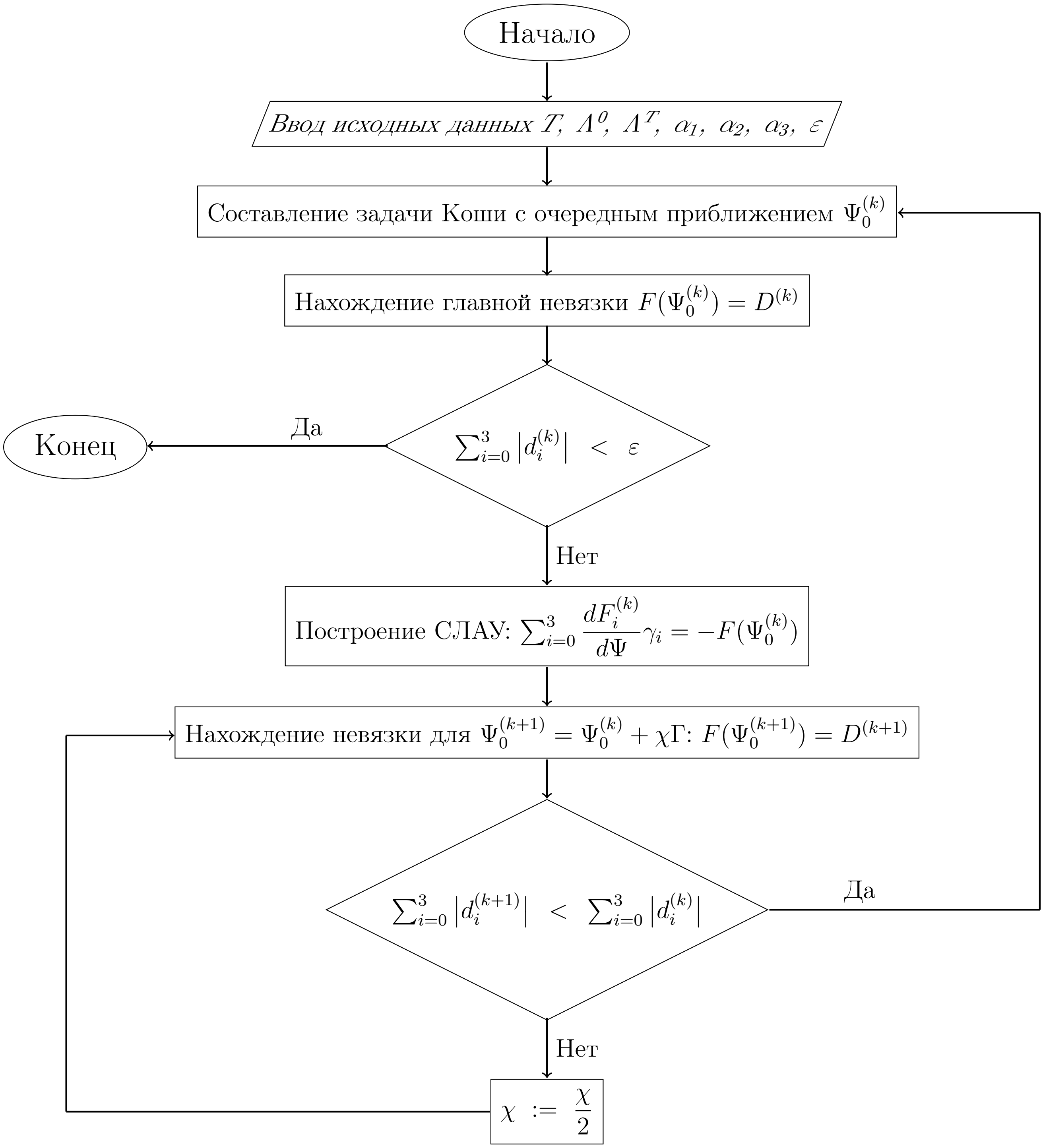


Рисунок 6.1 — Блок — схема алгоритма решения

Для решения краевой задачи была написана программа на языке Java [18], код которой помещен в приложение. Для построения графиков решений был использован язык R [20].

7 Исследование решений при малых углах поворота

7.1 Разные временные отрезки

Рассмотрим задачу оптимального управления для уравнения (4.1), где начальное положение твердого тела задано следующими углами Эйлера

$$\alpha = 10^\circ,\ \beta = 8^\circ,\ \gamma = 5^\circ, \tag{7.1}$$

где α — угол прецессии, β — угол нутации, γ — угол собственного вращения.

Углам в (7.1) соответствует кватернион

$$\begin{cases} \mathbf{\Lambda^0} &= \mathbf{\Lambda^0}(\lambda_0^0,\ (\lambda_1^0,\ \lambda_2^0,\ \lambda_3^0)), \\ \lambda_0^0 &= 0.9930873627220702, \\ \lambda_1^0 &= 0.0732173086418921, \\ \lambda_2^0 &= 0.037273661348003334, \\ \lambda_3^0 &= 0.083829528727505. \end{cases} \tag{7.2}$$

Конечное положение тела задано углами

$$\widetilde{\alpha} = 0^\circ,\ \widetilde{\beta} = 0^\circ,\ \widetilde{\gamma} = 0^\circ, \tag{7.3}$$

что соответствует следующему кватерниону

$$\begin{cases} \mathbf{\Lambda^T} &= \mathbf{\Lambda^T}(\lambda_0^T,\ (\lambda_1^T,\ \lambda_2^T,\ \lambda_3^T)), \\ \lambda_0^T &= 1,\ \lambda_1^T = 0,\ \lambda_2^T = 0,\ \lambda_3^T = 0. \end{cases} \tag{7.4}$$

Пусть требуется решить задачу с точностью $\varepsilon = 10^{-9}$ при фиксированных $\alpha_1 = 1000,\ \alpha_2 = 2000,\ \alpha_3 = 3000$.

Значения функционала качества для разных временных отрезков $[0,\ T]$, полученные с помощью программы, представлены в таблице 7.1.

Таблица 7.1

$T,\ c$	I
200	0.6665346560157165
210	0.6347947090668907
220	0.6059402291411088
230	0.5795948495036541
240	0.5554449303187555
250	0.5332270138435539
260	0.512718177373356
270	0.4937285213805331
280	0.47609527601585777
290	0.45967812105185146
300	0.44435544985946895
310	0.4300213419788282
320	0.4165831192013803
330	0.4039593366750249
340	0.39207813322040375
350	0.3808758580825576
360	0.3702959339180532
370	0.3602878996580692
380	0.35080660573398065
390	0.34181153391470087
400	0.3332662171330903

Закон изменения значений функционала качества при разных временных отрезках отражена на рисунке 7.1.

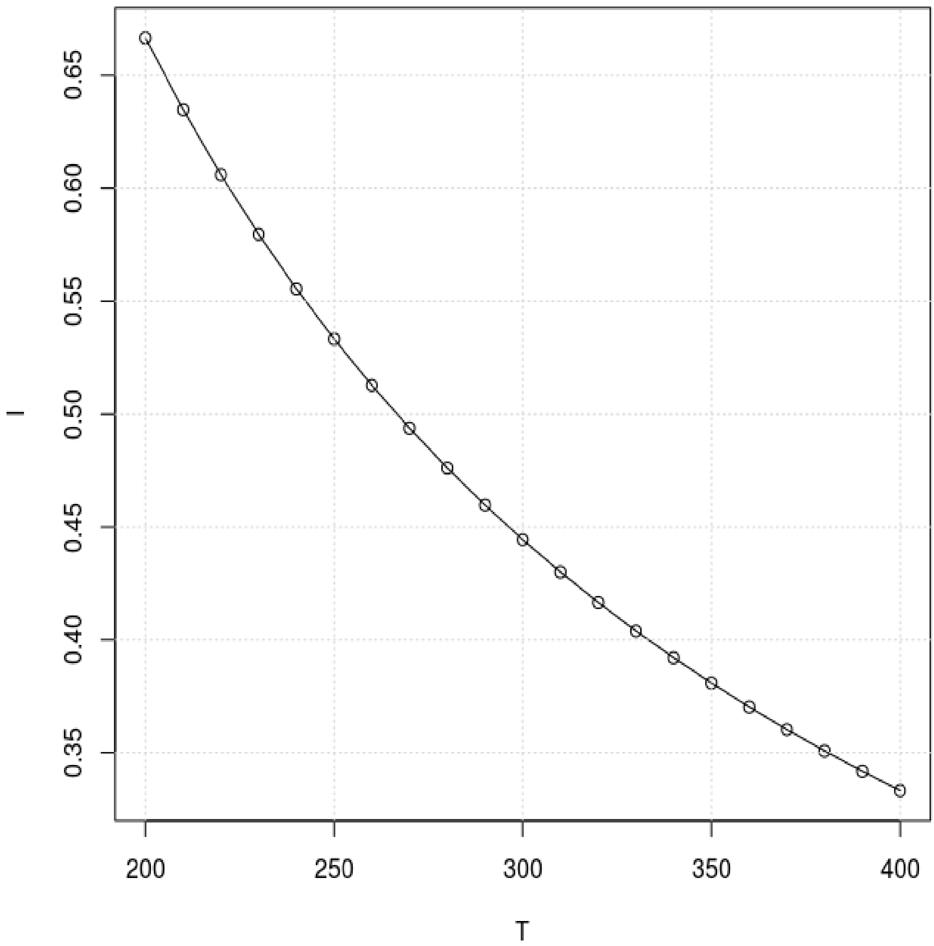


Рисунок 7.1 — закон изменения значений функционала качества при разных временных отрезках.

Из рисунка 7.1 можно делать вывод, что при увеличении времени T количество энергии I, которое требуется затратить на управление, уменьшается. Это легко объясняется тем, что при увелечении затрат энергии на управление увеличивается угловая скорость, и следовательно меньше времени затрачивается на поворот.

7.2 Разные весовые множители функционала качества

Рассмотрим теперь случаи при одинаковом параметре $T = 300c$ и разных значениях весовых множителей $\alpha_1, \ \alpha_2, \ \alpha_3$. Начальное положение тела также задано углами (7.1).

1) Зафиксируем значения двух весовых множителей и будем варьировать значением третьего множителя. Пусть $\alpha_1 = 1000$, $\alpha_2 = 2000$, α_3 пробегает по значениям: 1000, 2000, ... 10000.

Таблица 7.2

α_1	α_2	α_3	I
1000	2000	1000	0.2561562129231243
1000	2000	2000	0.3504523223918996
1000	2000	3000	0.44435544985946895
1000	2000	4000	0.5378731541347116
1000	2000	5000	0.631005289929338
1000	2000	6000	0.7237495719395564
1000	2000	7000	0.8161023547512606
1000	2000	8000	0.9080588213038621
1000	2000	9000	0.9996130262325005
1000	2000	10000	1.0907578924280608

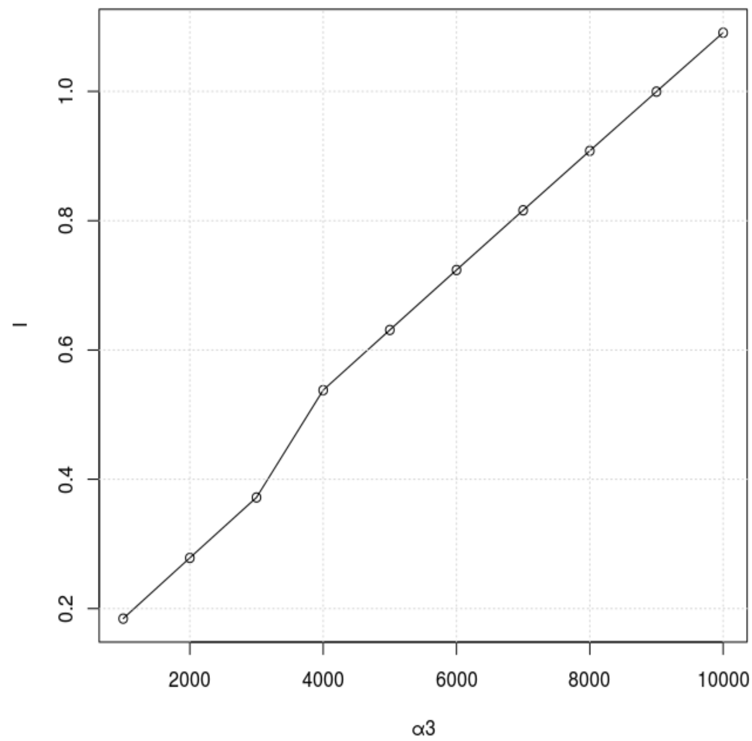


Рисунок 7.2 — закон изменения значений функционала качества при $\alpha_1 = 1000$, $\alpha_2 = 2000$.

Из графика на рисунке 7.2 видно, что между изменяемым параметром α_3 и значением величины функционала качества I существует линейная зависимость.

2) Теперь зафиксируем значение одного весового множителя и будем варьировать одинаковым образом значения двух остальных множителей:

Таблица 7.3

α_1	α_2	α_3	I
1000	1000	1000	0.18455058244466496
1000	2000	2000	0.3504523223918996
1000	3000	3000	0.5163281860856808
1000	4000	4000	0.6821975645883387
1000	5000	5000	0.8480643431827684
1000	6000	6000	1.0139298219178696
1000	7000	7000	1.1797945575930033
1000	8000	8000	1.3456588291157578
1000	9000	9000	1.5115227892739374
1000	10000	10000	1.6773865337492355

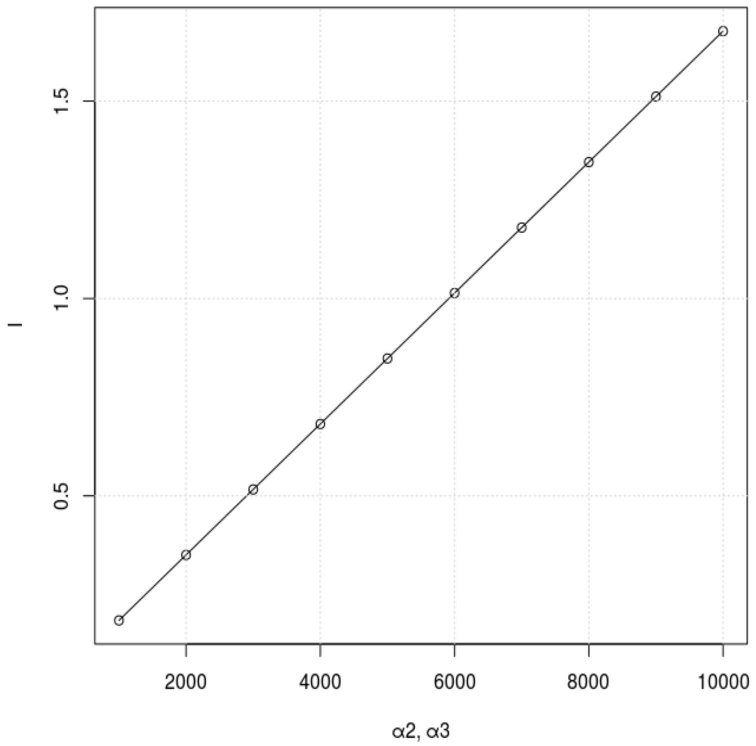


Рисунок 7.3 — закон изменения значений функционала качества при $\alpha_1 = 1000$.

Как и в первом случае, здесь линейная зависимость между изменяемыми параметрами α_1, α_2 и I.

3) Рассмотрим, наконец, случай, в котором все весовые множители варьируются одинаковым образом

Таблица 7.4

α_1	α_2	α_3	I
1000	1000	1000	0.18455058244466496
2000	2000	2000	0.3691011649278124
3000	3000	3000	0.5536517473623023
4000	4000	4000	0.7382023299085259
5000	5000	5000	0.9227529126527748
6000	6000	6000	1.1073034950203156
7000	7000	7000	1.2918540774575527
8000	8000	8000	1.4764046594125089
9000	9000	9000	1.660955240413333
10000	10000	10000	1.8455058248796425

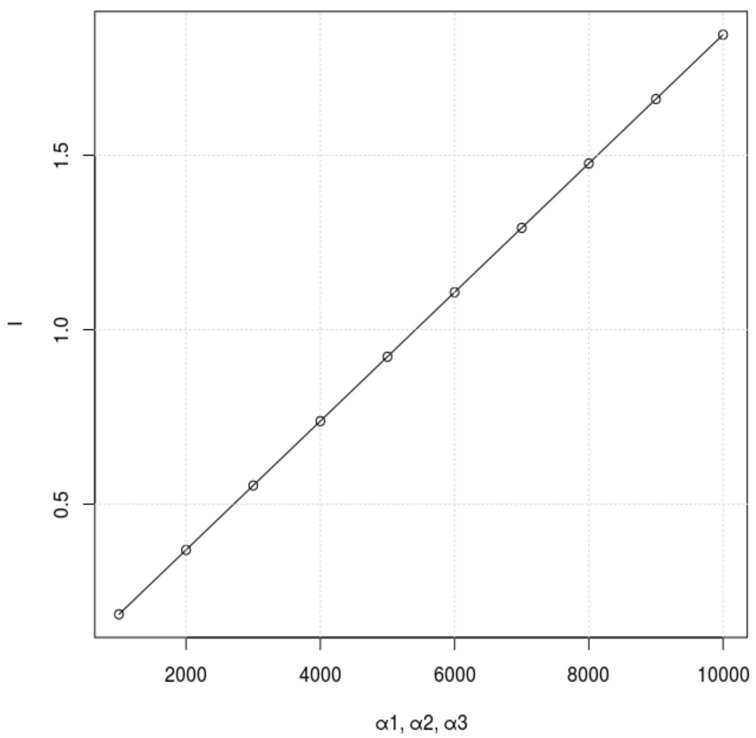


Рисунок 7.4 — закон изменения значений функционала качества при разных значениях весовых множителей.

Таким образом, можно сделать вывод из графиков 7.2 – 7.4, что при увеличении весовых множителей функционала качества энергетические затраты на поворот тела растут, причем зависимость этого роста линейная.

7.3 Разные начальные углы поворота

В двух первых случаях рассматривалась зависимость значений функционала качества и его параметров при некотором фиксированном начальном положении тела. Рассмотрим теперь значения функционала качества при разных начальных положениях твердого тела. Для этого зафиксируем конечное положение тела, которое задано углами Эйлера: $\widetilde{\alpha} = 0°$, $\widetilde{\beta} = 0°$, $\widetilde{\gamma} = 0°$, весовые множители $\alpha_i, i = \overline{1,3}$, где $\alpha_1 = 1000$, $\alpha_2 = 2000$, $\alpha_3 = 3000$ и время $T = 300c$. Начальное положение тела будем менять в соответствии с таблицей 7.5. В этой же таблице даны соответствующие значения функционала качества.

Таблица 7.5

α	β	γ	I
0°	0°	0°	0.0
1°	1°	1°	0.006056692698154153
2°	2°	2°	0.02408215835432236
3°	3°	3°	0.05385534350818591
4°	4°	4°	0.09514985533820493
5°	5°	5°	0.1477341814913685
6°	6°	6°	0.21137190897814612
7°	7°	7°	0.28582194497504043
8°	8°	8°	0.3708387365805886
9°	9°	9°	0.46617248981673004
10°	10°	10°	0.5715693910515096
11°	11°	11°	0.6867718203349862
12°	12°	12°	0.8115185700197344
13°	13°	13°	0.9455450560043213

продолжение таблицы 7.5			
α	β	γ	I
14°	14°	14°	1.0885835236291768
15°	15°	15°	1.2403632727704637
16°	16°	16°	1.4006108189429844
17°	17°	17°	1.5690501304510134
18°	18°	18°	1.7454027952571545
19°	19°	19°	1.929388211767391
20°	20°	20°	2.1207237623553117
21°	21°	21°	2.319124980674097
22°	22°	22°	2.5243057234840194
23°	23°	23°	2.735978298083406
24°	24°	24°	2.9538536340429173
25°	25°	25°	3.1776413757372017
26°	26°	26°	3.4070500639069183
27°	27°	27°	3.641787193331258

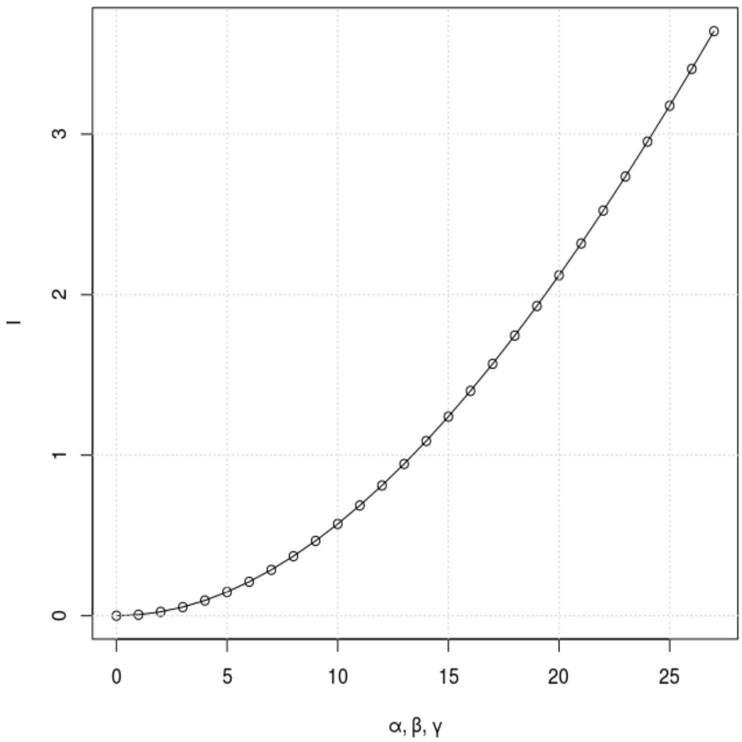


Рисунок 7.5 — закон изменения значений функционала качества при разных начальных углов поворота.

График на рисунке 7.5 показывает очевидную закономерность — чем больше начальное отклонение тела от конечного его положения, тем больше требуется энергетических затрат на управление.

Таким образом, в пунктах 7.1 – 7.3 были рассмотрены основные закономерности поставленной задачи оптимального управления (4.1) – (4.4) для функционала качества (4.5) при различных параметрах задачи путем анализа решения соответствующей краевой задачи (5.12). Несмотря на то, что характер большинства этих зависимостей можно определить из вида самого функционала качества, это позволяет лишний раз убедиться в эффективности выбранных методов для решения задачи, а также корректности программной реализации.

8 Примеры для больших углов поворота

Расмотрим задачу оптимального управления, которой соответствует краевая задача (8.1) для тела, начальное положение которого задано углами Эйлера: $\alpha = -78.4°$, $\beta = -39.9°$, $\gamma = 112.9°$, а конечное — $\widetilde{\alpha} = 0°$, $\widetilde{\beta} = 0°$, $\widetilde{\gamma} = 0°$. Пусть требуется решить задачу с точностью $\varepsilon = 10^{-9}$ при весовых множителях $\alpha_1 = 1000$, $\alpha_2 = 2000$, $\alpha_3 = 3000$ для времени $T = 300c$.

$$\begin{cases} 2\dot{\mathbf{\Lambda}} = \mathbf{\Lambda} \circ \mathbf{\Omega}, \\ \mathbf{\Lambda}(0) = \mathbf{\Lambda^0}(\lambda_0^0,\ (\lambda_1^0,\ \lambda_2^0,\ \lambda_3^0)), \\ \lambda_0^0 = -0.5821271946729387, \\ \lambda_1^0 = 0.10821947847990215, \\ \lambda_2^0 = 0.641192910029563, \\ \lambda_3^0 = 0.48814764756943485. \\ \mathbf{\Lambda}(T) = \mathbf{\Lambda^{\boldsymbol{T}}}(\lambda_0^T,\ (\lambda_1^T,\ \lambda_2^T,\ \lambda_3^T)), \\ \lambda_0^T = 1,\ \lambda_1^T = 0,\ \lambda_2^T = 0,\ \lambda_3^T = 0. \end{cases} \tag{8.1}$$

График изменения компонент кватерниона $\mathbf{\Lambda} = \mathbf{\Lambda}(\lambda_0,\ (\lambda_1,\ \lambda_2, \lambda_3))$ с течением времени, найденный в ходе решения, представлен на рисунке 8.1. По данному рисунку видно, что оптимальное управление переводит тело из заданного начального углового положения в требуемое конечное.

Найденное решение $\mathbf{\Lambda}(t)$ позволяет получить данные об изменениях углов Эйлера, задающие угловое положение тела. На рисунках 8.2 $-$ 8.4 представлены изменения углов прецессии, нутации, собственного вращения соответственно.

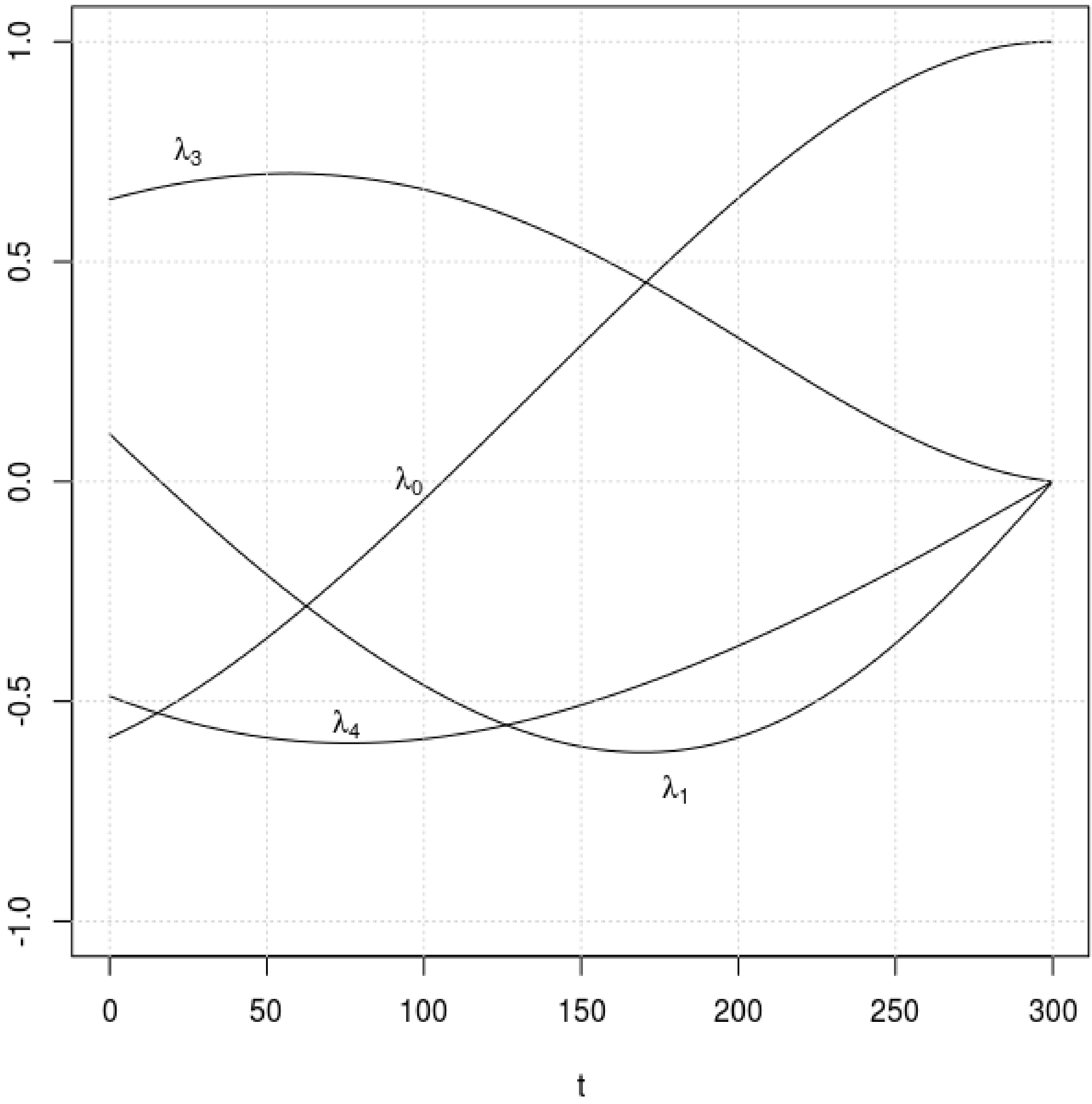


Рисунок 8.1 — график изменения компонент оптимальной траектории.

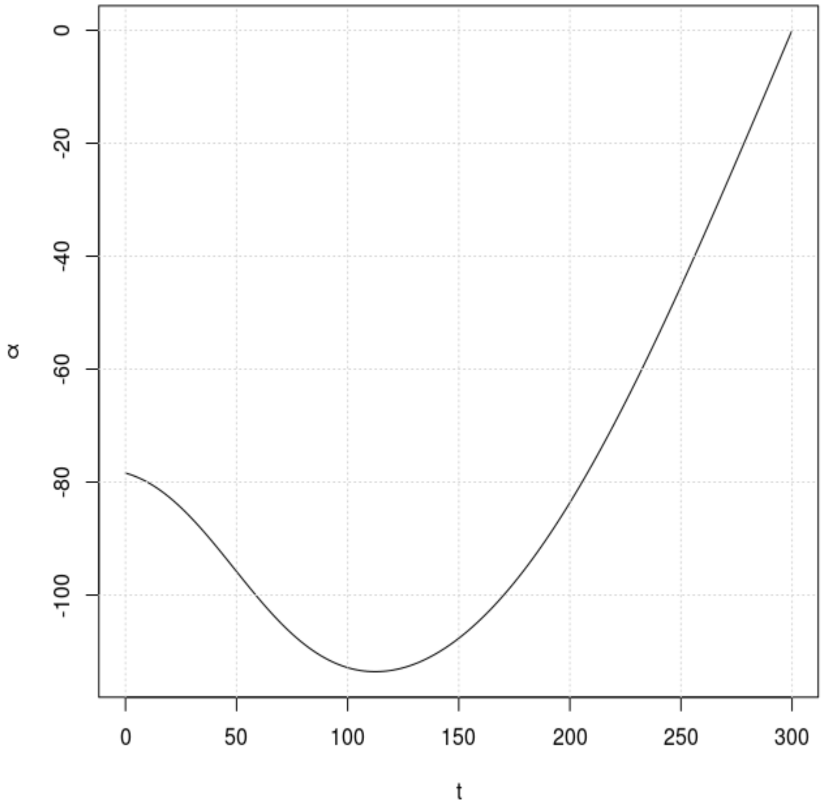


Рисунок 8.2 — изменение угла прецессии.

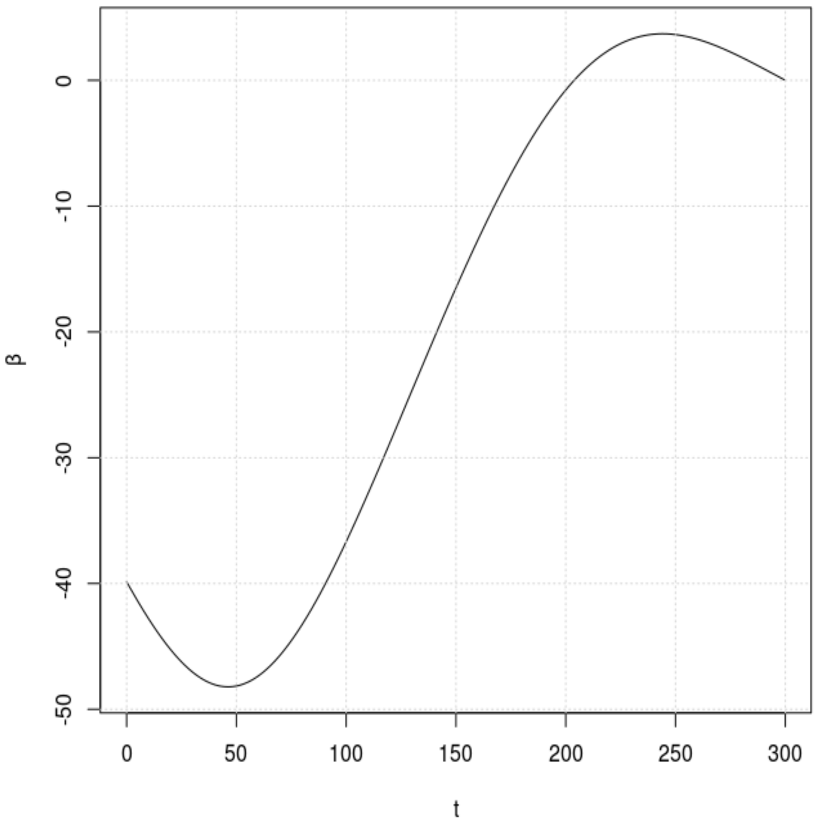


Рисунок 8.3 — изменение угла нутации.

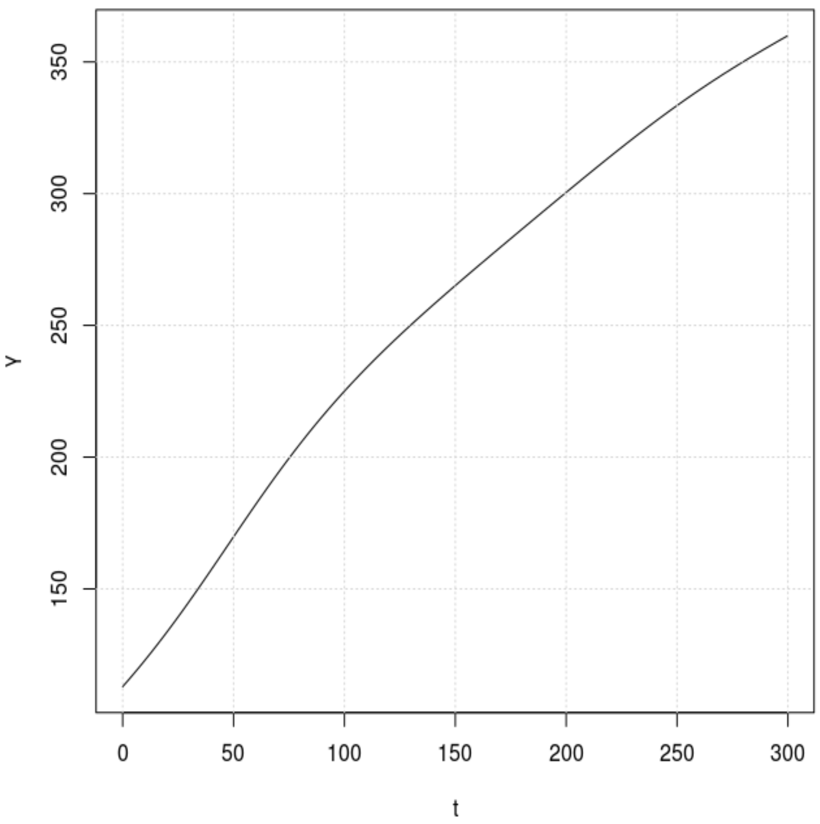


Рисунок 8.4 — изменение угла собственного вращения.

Рисунки 8.2 – 8.4 полностью соответствуют рисунку 8.1, и следовательно требованиям для оптимального управления.

В поставленной задаче в качестве оптимального управления, как было сказано ранее, выступает угловая скорость $\mathbf{\Omega} = \mathbf{\Omega}(\omega_0,\ (\omega_1,\ \omega_2,\ \omega_3))$, которая также находится в ходе решения краевой задачи (5.12). График изменения компонент угловой скорости $\omega_1,\ \omega_2,\ \omega_3$ представлен на рисунке 8.5.

Основная трудность, с которой можно столкнуться при решении задачи оптмального управления для больших углов отклонения между начальным и конечным положениями тела — это нахождение начального приближения Ψ, поэтому желательно знать, где приблизительно его нужно искать.

Для данной задачи график изменения компонент кватерниона $\mathbf{\Psi} = \mathbf{\Psi}(\psi_0,\ (\psi_1,\ \psi_2, \psi_3))$, который получается на последней итерации метода Ньютона решения краевой задачи (5.12), представлены на рисунке 8.6.

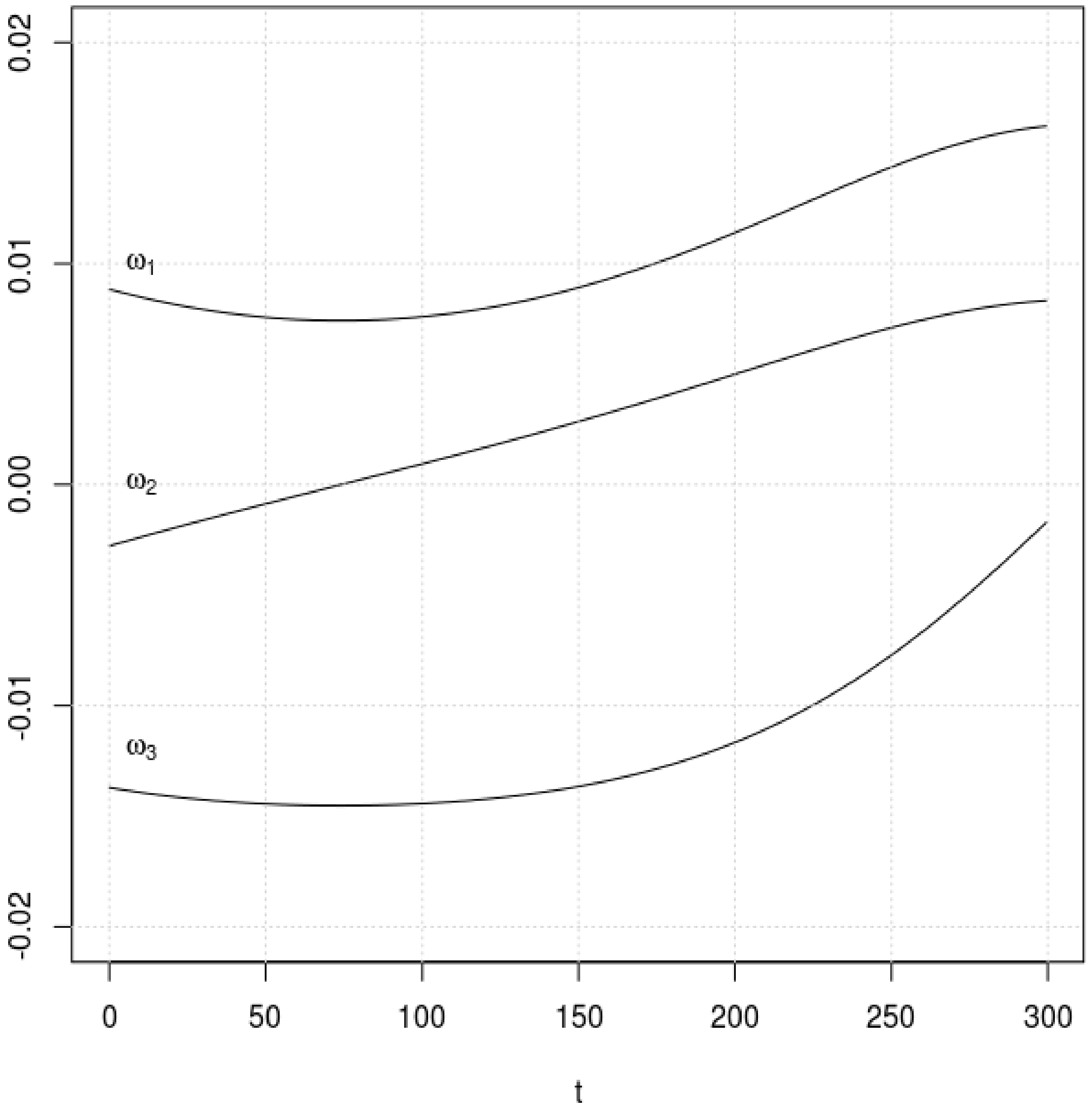


Рисунок 8.5 — график изменения компонент оптимального управления.

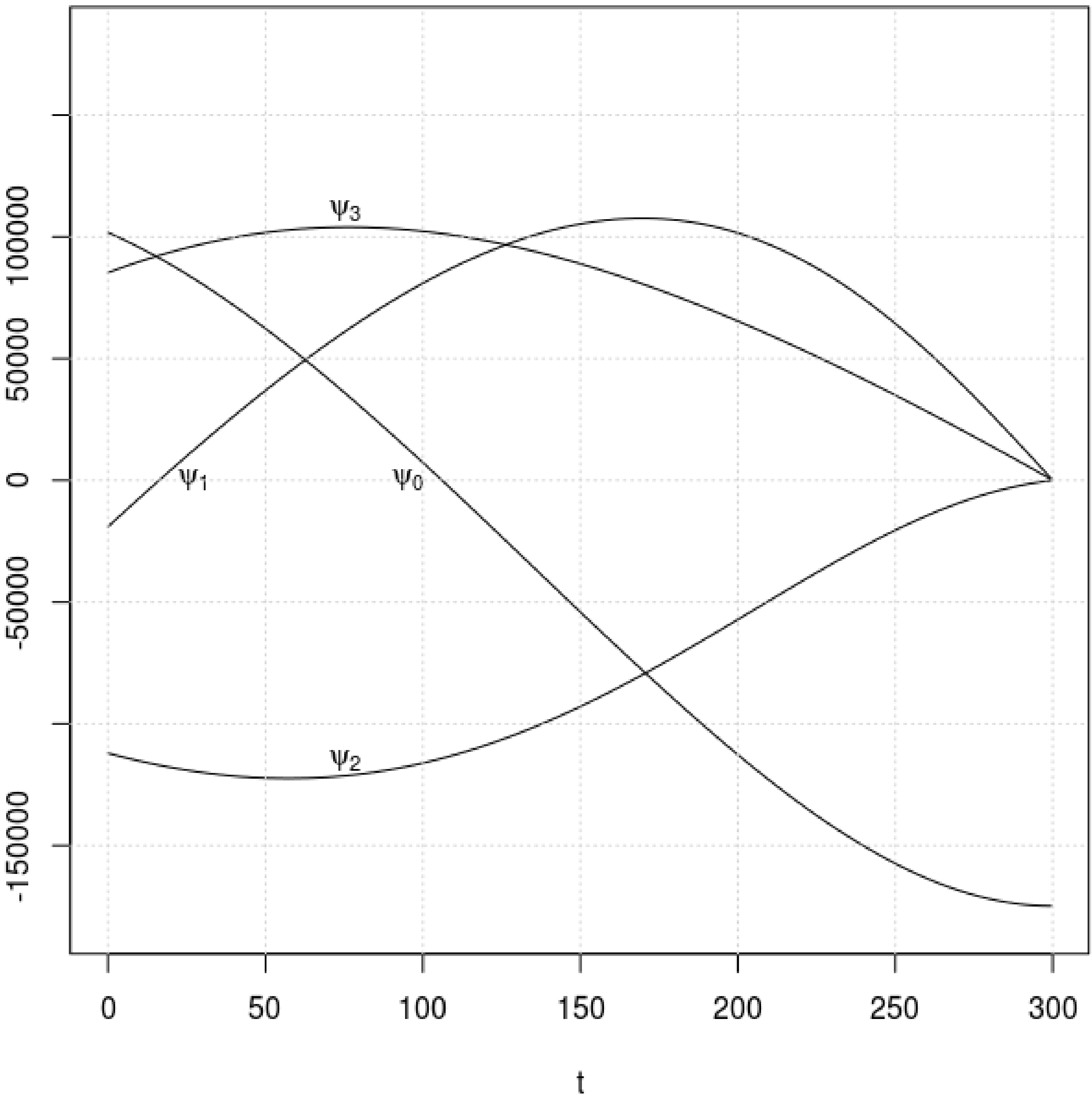


Рисунок 8.6 — график изменения компонент кватерниона Ψ.

Рассмотрим теперь задачу (8.1) при $T = 301c$, $T = 302c$. Значения функционала качества при таких параметрах T, включая случай $T = 300c$ представлены в таблице 8.1.

Таблица 8.1

T, с	I
300	143.0483289121585
301	142.5730831794837
302	142.10098482727184

Графическое представление таблицы 8.1 отражено на рисунке 8.7, из которого следует, что при увеличении параметра T энергетические затраты на управление уменьшаются, что соответствует выводам, сделанных для малых углов отклонения.

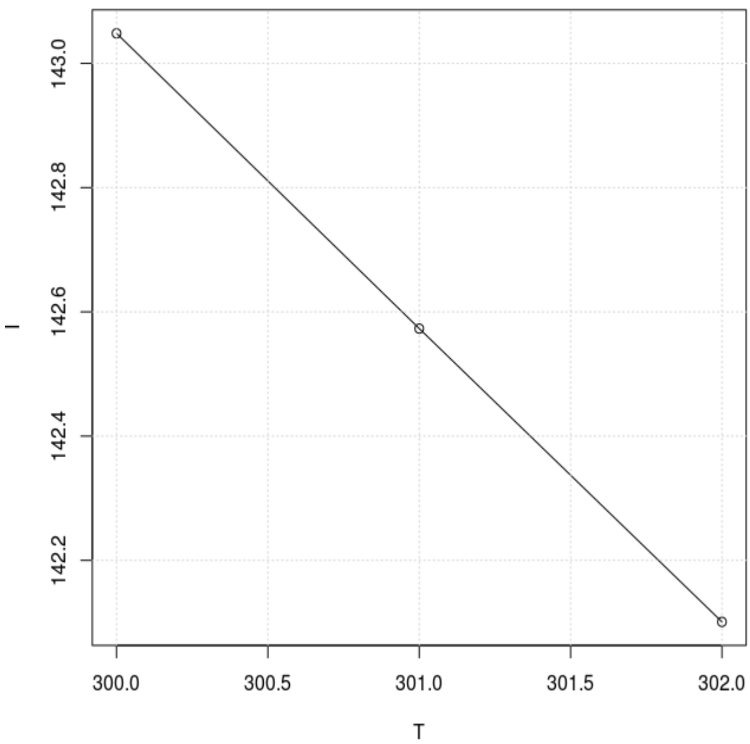


Рисунок 8.7 — графическое представление таблицы 8.1.

ЗАКЛЮЧЕНИЕ

В предоставленной квалификационной работе удалось решить задачу оптимального управления углового движения ИСЗ, для которой требовалось составить и решить краевую задачу с помошью принципа максимума Л.С. Понтрягина.

Были рассмотрены различные поведения системы при различных параметрах. Также удалось программно реализовать алгоритм численного решения задачи с применением алгебры кватернионов.

Полученные в работе теоретические результаты позволяют понять основные зависимости функционала качества управления и его параметров, а также прогнозировать поведение системы при изменениях параметров самой задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Челноков, Ю. Н. *Кватернионные и бикватернионные модели и методы механики твёрдого тела и их приложения.* М.: Физматлит, 2006. – 512с.

2. Бранец, В. Н., and Шмыглевский, И. П. *Применение кватернионов в задачах ориентации твердого тела.* М.: Наука, 1973. – 320с.

3. Понтрягин, Л. С., and Болтянский, В. Г. and Гамкрелидзе, Р. В. and Мищенко, Е. Ф. *Математическая теория оптимальных процессов.* М.: Наука, 1983 – 393с.

4. Сапунков, Я. Г. *Численное исследование систем автоматического управления.* М.: Наука, 2001. – 24с.

5. Горелов Ю.Н. *Численные методы решения обыкновенных дифференциальных уравнений (Метод Рунге - Кутта).* Изд-во «Самарский университет», 2006. – 48 с.

6. Антипова А.С., Бирюков Б.Г. *Аналитическое и численное исследование кинематической задачи оптимальной переориентации твердого тела.* УДК – 48 с.

7. Асланов. В. С. *Динамика твёрдого тела и систем тел.* Самарский государственный аэрокосмический университет, 2011 – 216с.

8. Тарасов В.Н., Бахарева Н.Ф. *Численные методы. Теория, алгоритмы, программы..* Оренбург: ИПК ОГУ, 2008. – 264 с.

9. Ермолин В. С., Королев В. С., Потоцкая Е. Ю. *Теоретическая механика. Часть I. Кинематика. Учебное пособие..* СПб: СПбГУ, ВВМ, 2013.— 225 с.

10. Теляковский С. А. *Курс лекций по математическому анализу.* М.: МИАН, 2009. – 212 с.

11. Пантелеев А.В., Бортаковский А.С., Летова Т.А. *Оптимальное управление в примерах и задачах..* М: Издательство МАИ, 1996. — 583 с.

12. Knuth: Computers and Typesetting,
http://www-cs-faculty.stanford.edu/~uno/abcde.html (дата последнего обращения: 10.05.16)

13. Прямые методы решения линейных систем
http://www.math.spbu.ru/user/pan/Page11-gauss.pdf (дата последнего обращения: 10.05.16)

14. Метод Гаусса
http://pedsovet.info/info/pages/referats/info_00036.htm (дата последнего обращения: 10.05.16)

15. Углы Эйлера
https://ru.wikipedia.org/wiki/%D0%A3%D0%B3%D0%BB%D1%8B_%D0%AD%D0%B9%D0%BB%D0%B5%D1%80%D0%B0
(дата последнего обращения: 10.05.16)

16. Conversion between quaternions and Euler angles
http://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles
(дата последнего обращения: 10.05.16)

17. Newton's method
https://en.wikipedia.org/wiki/Newton%27s_method (дата последнего обращения: 10.05.16)

18. Java Platform, Standard Edition (Java SE) 8
https://docs.oracle.com/javase/8/ (дата последнего обращения: 10.05.16)

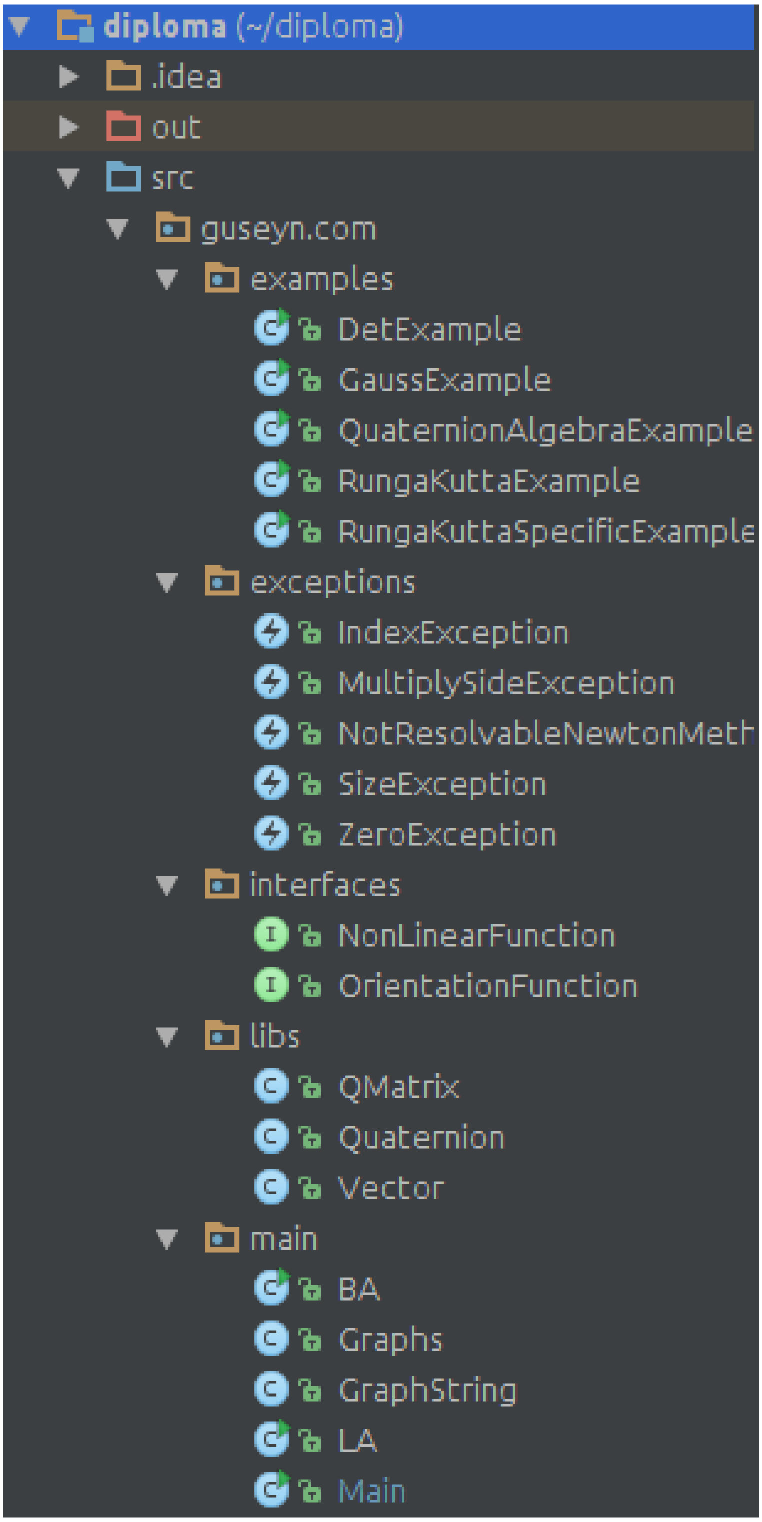
19. Java: What Is an Interface?
https://docs.oracle.com/javase/tutorial/java/concepts/interface.html (дата последнего обращения: 10.05.16)

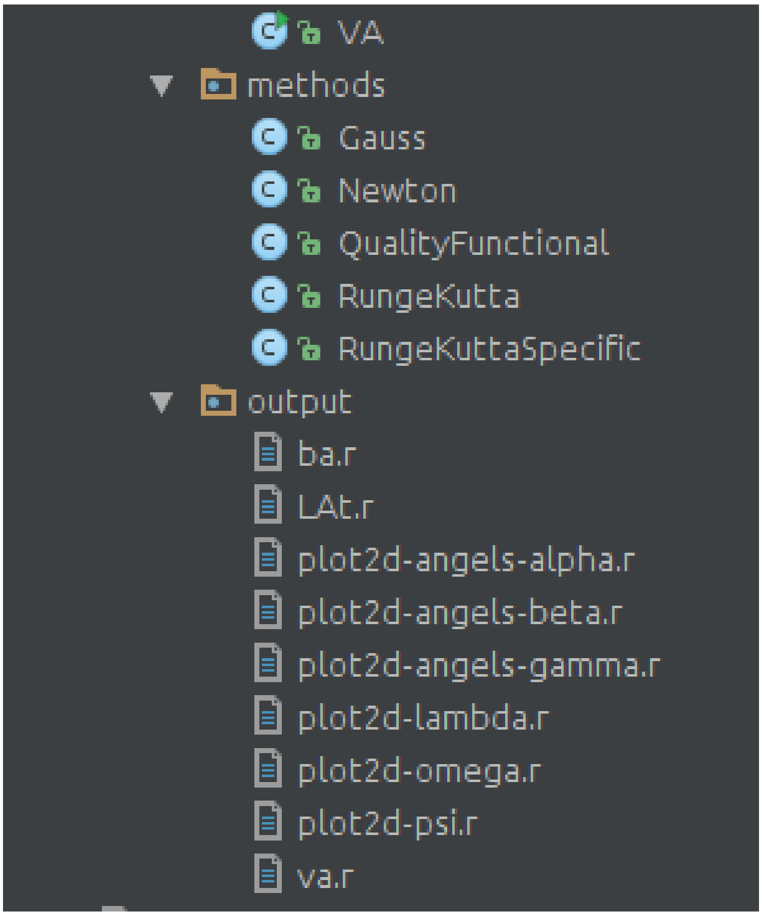
20. R: Documentation
https://www.r-project.org/other-docs.html (дата последнего обращения: 10.05.16)

21. Block diagram
https://en.wikipedia.org/wiki/Block_diagram (дата последнего обращения: 10.05.16)

ПРИЛОЖЕНИЕ А

Структура программы




VA
methods
Gauss
Newton
QualityFunctional
RungeKutta
RungeKuttaSpecific
output
ba.r
LAt.r
plot2d-angels-alpha.r
plot2d-angels-beta.r
plot2d-angels-gamma.r
plot2d-lambda.r
plot2d-omega.r
plot2d-psi.r
va.r

ПРИЛОЖЕНИЕ Б

Исходный код программы

Listing Б.1 — Quaternion.java

```java
package guseyn.com.libs;

import guseyn.com.exceptions.IndexException;
import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;

/*            -                         */

public class Quaternion {

    private double scalar;
    private Vector vector;
    public final static int SIZE = 4;
    public final static int VECTOR_SIZE = 3;

    public Quaternion(Quaternion quaternion) {
        this.scalar = quaternion.getScalar();
        this.vector = new Vector(quaternion.getVector());
    }

    public Quaternion(double scalar, Vector vector) {
        this.scalar = scalar;
        this.vector = new Vector(vector);
    }

    public Quaternion getPairingQuaternion() {
        return new Quaternion(scalar, vector.multiplyWithScalar(-1));
    }

    public double getNorm() {
        return Math.sqrt(Math.pow(scalar, 2) +
                Math.pow(getICoefficient(), 2) +
                Math.pow(getJCoefficient(), 2) +
                Math.pow(getKCoefficient(), 2));
    }

    public Quaternion plus(Quaternion quaternion) throws SizeException {
        return new Quaternion(scalar + quaternion.scalar,
                vector.plus(quaternion.vector)
        );
    }

```

```
    public Quaternion minus(Quaternion quaternion) throws SizeException {
        return this.plus(quaternion.multiplyWithScalar(-1));
    }

    public Quaternion multiplyWithScalar(double s) {
        return new Quaternion(scalar * s, vector.multiplyWithScalar(s));
    }

    public Quaternion multiplyWithQuaternion(Quaternion quaternion, String side) throws SizeException,
            MultiplySideException {
        if (quaternion.getVector().getSize() == VECTOR_SIZE) {
            double s = scalar * quaternion.scalar -
                    vector.scalarMultiplyWithVector(quaternion.vector);
            Vector v = quaternion.getVector().
                    multiplyWithScalar(scalar).
                    plus(vector.multiplyWithScalar(quaternion.scalar)).
                    plus(vector.vectorMultiplyWith3DVector(quaternion.vector, side));
            return new Quaternion(s, v);
        } else {
            throw new SizeException("quaternionMultiplication size e");
        }
    }

    public Quaternion normalize() {
        return this.multiplyWithScalar(1.0 / getNorm());
    }

    public Quaternion getReverseQuaternion() {
        double s = 1.0 / Math.pow(getNorm(), 2);
        return getPairingQuaternion().multiplyWithScalar(s);
    }

    public static void transpose(Quaternion[] quaternions) throws IndexException {
        for (int i = 0; i < SIZE; i++) {
            for (int j = i; j < SIZE; j++) {
                double tmp = quaternions[i].get(j);
                quaternions[i].set(j, quaternions[j].get(i));
                quaternions[j].set(i, tmp);
            }
        }
    }

```

```java
    public static Quaternion getFromEulerAngles(double phi, double teta, double psi) {
        phi = Math.toRadians(phi) / 2;
        teta = Math.toRadians(teta) / 2;
        psi = Math.toRadians(psi) / 2;
        return new Quaternion(Math.cos(phi) * Math.cos(teta) * Math.cos(psi)
                + Math.sin(phi) * Math.sin(teta) * Math.sin(psi),
                new Vector(Math.sin(phi) * Math.cos(teta) * Math.cos(psi) -
                        Math.cos(phi) * Math.sin(teta) * Math.sin(psi),
                        Math.cos(phi) * Math.sin(teta) * Math.cos(psi) +
                            Math.sin(phi) * Math.cos(teta) * Math.sin(psi),
                        Math.cos(phi) * Math.cos(teta) * Math.sin(psi) -
                            Math.sin(phi) * Math.sin(teta) * Math.cos(psi)));
    }

    public static Double[] getEulerAngels(Quaternion q) throws IndexException {
        double phi = Math.toDegrees(Math.atan2(2 * (q.get(0) * q.get(1) + q.get(2) * q.get(3)),
                (1 - 2 * (Math.pow(q.get(1), 2) + Math.pow(q.get(2), 2)))));
        double teta = Math.toDegrees(Math.asin(2 * (q.get(0) * q.get(2) - q.get(3) * q.get(1))));
        double psi = Math.toDegrees(Math.atan2(2 * (q.get(0) * q.get(3) + q.get(1) * q.get(2)),
                (1 - 2 * (Math.pow(q.get(2), 2) + Math.pow(q.get(3), 2)))));
        return new Double[]{phi, teta, psi};
    }

    public static Quaternion clone(Quaternion q) {
        return new Quaternion(q.scalar,
                new Vector(q.getICoefficient(), q.getJCoefficient(), q.getKCoefficient()));
    }

    public static Quaternion[] cloneArray(Quaternion[] q) {
        Quaternion[] q2 = new Quaternion[q.length];
        for (int i = 0; i < q.length; i++) {
            q2[i] = Quaternion.clone(q[i]);
        }
        return q2;
    }
```

```java

    public double get(int index) throws IndexException {
        if (index == 0) {
            return getScalar();
        } else if (index > 0 && index < SIZE) {
            return vector.getCoordinate(index - 1);
        } else {
            throw new IndexException("Quaternion index is over than
               3");
        }
    }

    public void set(int index, double value) throws IndexException {
        if (index == 0) {
            scalar = value;
        } else if (index > 0 && index < SIZE) {
            vector.setCoordinate(index - 1, value);
        } else {
            throw new IndexException("Quaternion index is over than
               3");
        }
    }

    public double getICoefficient() {
        return vector.getCoordinate(0);
    }

    public double getJCoefficient() {
        return vector.getCoordinate(1);
    }

    public double getKCoefficient() {
        return vector.getCoordinate(2);
    }

    public double getScalar() {
        return scalar;
    }

    public Vector getVector() {
        return vector;
    }

    @Override
    public String toString() {
        return "Quaternion{" +
                "scalar=" + scalar +
                ", vector=" + vector +
                "} \n";
    }
```

```
}
```

Listing Б.2 — Vector.java

```
package guseyn.com.libs;

import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;

import java.util.Arrays;

public class Vector {

    private int size;
    private double[] coordinates;

    public Vector(double... values) {
        this.size = values.length;
        this.coordinates = values;
    }

    public Vector(Vector vector) {
        this.size = vector.size;
        this.coordinates = vector.coordinates;
    }

    public Vector plus(Vector vector) throws SizeException {
        double[] values = new double[size];
        if (size == vector.size) {
            for (int i = 0; i < size; i++) {
                values[i] = coordinates[i] + vector.getCoordinate(i)
                    ;
            }
        } else {
            throw new SizeException("plus size e");
        }
        return new Vector(values);
    }

    public Vector minus(Vector vector) throws SizeException {
        return this.plus(vector.multiplyWithScalar(-1));
    }

    public Vector multiplyWithScalar(double scalar) {
        double[] values = new double[size];
        for (int i = 0; i < size; i++) {
            values[i] = coordinates[i] * scalar;
        }
        return new Vector(values);
    }
```

```java

    public double scalarMultiplyWithVector(Vector vector) throws SizeException {
        double res = 0;
        if (size == vector.size) {
            for (int i = 0; i < size; i++) {
                res += coordinates[i] * vector.getCoordinate(i);
            }
        } else {
            throw new SizeException("scalarMultiplyWithVector size e");
        }
        return res;
    }

    public Vector vectorMultiplyWith3DVector(Vector vector, String side) throws SizeException,

        if (size == 3) {
            double c0 = vector.getCoordinate(0);
            double c1 = vector.getCoordinate(1);
            double c2 = vector.getCoordinate(2);
            boolean isLeft = side.equals("left");
            boolean isRight = side.equals("right");
            Vector result = new Vector(
                    coordinates[1] * c2 - coordinates[2] * c1,
                    coordinates[2] * c0 - coordinates[0] * c2,
                    coordinates[0] * c1 - coordinates[1] * c0
            );
            if (isLeft) {
                return result.multiplyWithScalar(-1);
            } else if (isRight) {
                return result;
            } else {
                throw new MultiplySideException("vectorMultiplyWith3DVector multiply side e");
            }
        } else {
            throw new SizeException("vectorMultiplyWith3DVector size e");
        }
    }

    public double getDescartesLength() {
        double length = 0;
        for (int i = 0; i < size; i++) {
            length += Math.pow(getCoordinate(i), 2);
```

```java
        }
        return Math.sqrt(length);
    }

    public Vector normalize() {
        return this.multiplyWithScalar(1.0 / getDescartesLength());
    }

    public Quaternion getRotationQuaternion(double angle) throws
       SizeException {
        double radian = Math.PI / 180;
        angle *= radian;
        if (size == 3) {
            double halfAngleSin = Math.sin(angle / 2);
            double halfAngleCos = Math.cos(angle / 2);
            return new Quaternion(halfAngleCos,
                    this.normalize().multiplyWithScalar(halfAngleSin
                       ));
        } else {
            throw new SizeException("getRotationQuaternion size e");
        }
    }

    public int getSize() {
        return size;
    }

    public double[] getCoordinates() {
        return coordinates;
    }

    public double getCoordinate(int position) {
        return coordinates[position];
    }

    public void setSize(int size) {
        this.size = size;
    }

    public void setCoordinates(double... coordinates) {
        this.coordinates = coordinates;
        this.size = coordinates.length;
    }

    public void setCoordinate(int index, double value) {
        this.coordinates[index] = value;
    }

    @Override
    public String toString() {
```

```
        return "Vector{" +
                "size=" + size +
                ", coordinates=" + Arrays.toString(coordinates) +
                '}';
    }
}
```

Listing Б.3 — QMatrix.java

```
package guseyn.com.libs;

import guseyn.com.exceptions.IndexException;
import guseyn.com.exceptions.SizeException;

public class QMatrix {

    private static final int SIZE3 = 3;
    private static final int SIZE4 = 4;

    public static double getDeterminant(Quaternion[] q) throws
       IndexException, SizeException {
        double[][] a = new double[SIZE4][SIZE4];
        for (int i = 0; i < Quaternion.SIZE; i++) {
            for (int j = 0; j < Quaternion.SIZE; j++) {
                a[i][j] = q[i].get(j);
            }
        }

        double result = 0;
        for (int i = 0; i < SIZE4; i++) {
            double[][] M = getM(a, SIZE4, i);
            double d = get3Determinant(M);
            double r = a[0][i] * d;
            result += (i % 2 == 0) ? r : -r;
        }
        return result;
    }

    private static double get3Determinant(double a[][]) {
        double result = 0;
        for (int i = 0; i < SIZE3; i++) {
            double[][] M = getM(a, SIZE3, i);
            double d = get2Determinant(M);
            double r = a[0][i] * d;
            result += (i % 2 == 0) ? r : -r;
        }

        return result;
    }

```

```java
    private static double get2Determinant(double[][] a) {
        return a[0][0] * a[1][1] - a[1][0] * a[0][1];
    }

    private static double[][] getM(double[][] a, int size, int index
       ) {
        double[][] d = new double[size - 1][size - 1];
        for (int j = 1, n = 0; j < size; j++, n++) {
            int m = 0;
            for (int k = 0; k < size; k++) {
                if (k != index) {
                    d[n][m] = a[j][k];
                    m++;
                }
            }
        }
        return d;
    }

}
```

Listing Б.4 — NonLinearFunction.java

```java
package guseyn.com.interfaces;

import guseyn.com.exceptions.IndexException;
import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;
import guseyn.com.exceptions.ZeroException;
import guseyn.com.libs.Quaternion;

@FunctionalInterface
public interface NonLinearFunction<Approximation, Solution> {

    public Solution apply(Approximation approximation) throws
       ZeroException, MultiplySideException, SizeException,
       IndexException;
}
```

Listing Б.5 — OrientationFunction.java

```java
package guseyn.com.interfaces;

import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;

@FunctionalInterface
public interface OrientationFunction< InputTime, InputQuaternion,
   OutputQuaternion> {
```

```java
    public OutputQuaternion apply(InputTime inputTime,
        InputQuaternion inputQuaternion) throws SizeException,
        MultiplySideException;
}
```

Listing Б.6 — IndexException.java

```java
package guseyn.com.exceptions;

public class IndexException extends Exception {

    public IndexException(String message) {
        super(message);
    }

}
```

Listing Б.7 — MultiplySideException.java

```java
package guseyn.com.exceptions;

public class MultiplySideException extends Exception {

    public MultiplySideException(String message) {
        super(message);
    }

}
```

Listing Б.8 — NotResolvableNewtonMethodException.java

```java
package guseyn.com.exceptions;

public class NotResolvableNewtonMethodException extends Exception {

    public NotResolvableNewtonMethodException(String message) {
        super(message);
    }
}
```

Listing Б.9 — SizeException.java

```java
package guseyn.com.exceptions;

public class SizeException extends Exception {

    public SizeException(String message) {
        super(message);
    }

}
```

Listing Б.10 — ZeroException.java

```java
package guseyn.com.exceptions;

public class ZeroException extends Exception {

    public ZeroException(String message) {
        super(message);
    }

}
```

Listing Б.11 — Gauss.java

```java
package guseyn.com.methods;

import guseyn.com.exceptions.IndexException;
import guseyn.com.exceptions.SizeException;
import guseyn.com.exceptions.ZeroException;
import guseyn.com.libs.Quaternion;
import guseyn.com.libs.Vector;

import java.util.Arrays;
import java.util.HashMap;

public class Gauss {

    private Quaternion[] a;
    private Quaternion b;
    int[] jmaxes = new int[Quaternion.SIZE];

    public Gauss(Quaternion[] a, Quaternion b) {
        this.a = Quaternion.cloneArray(a);
        this.b = Quaternion.clone(b);
    }

    public Quaternion solve() throws SizeException, IndexException,
       ZeroException {
        if (a.length == Quaternion.SIZE) {

            Quaternion.transpose(a);
            Quaternion solution = new Quaternion(0, new Vector(0, 0,
                0));

            for (int i = 0; i < Quaternion.SIZE; i++) {
                HashMap<String, Integer> maxIndexes = findMaxElement
                   (i);
                int imax = maxIndexes.get("i");
                int jmax = maxIndexes.get("j");
                swapTwoEquations(imax, i);
                for (int j = i + 1; j < Quaternion.SIZE; j++) {
```

```
                    double c = -1.0 * a[j].get(jmax) / a[i].get(jmax);
                    for (int k = 0; k < Quaternion.SIZE; k++) {
                        if (!Arrays.asList(jmaxes).contains(k)) {
                            a[j].set(k, a[j].get(k) + a[i].get(k) * c);
                            if (k == jmax) {
                                a[j].set(k, 0);
                            }
                        }
                    }
                    b.set(j, b.get(j) + b.get(i) * c);
                }
                jmaxes[i] = jmax;
            }

            Integer[] indexes = new Integer[Quaternion.SIZE];

            for (int n = Quaternion.SIZE - 1; n >= 0; n--) {
                double s = 0;
                double p = 0;
                int solutionIndex = 0;
                for (int m = 0; m < Quaternion.SIZE; m++) {
                    if (indexes[m] == null && a[n].get(m) != 0.0) {
                        indexes[m] = 1;
                        p = a[n].get(m);
                        solutionIndex = m;
                    } else if (indexes[m] != null && a[n].get(m) != 0) {
                        s += a[n].get(m) * solution.get(m);
                    }
                }
                solution.set(solutionIndex, (b.get(n) - s) / p);

            }

            return solution;

        } else {
            throw new SizeException("size of a is not 4");
        }
    }

    public static Quaternion checkB(Quaternion[] a, Quaternion solution) throws IndexException {
        Quaternion b = new Quaternion(0, new Vector(0, 0, 0));
        for (int i = 0; i < Quaternion.SIZE; i++) {
            double s = 0;
            for (int j = 0; j < Quaternion.SIZE; j++) {
                s += a[j].get(i) * solution.get(j);
```

```
            }
            b.set(i, s);
        }
        return b;
    }

    public HashMap<String, Integer> findMaxElement(int step) throws
        IndexException {
        int imax = step;
        int jmax = step;
        for (int i = step; i < Quaternion.SIZE; i++) {
            for (int j = 0; j < Quaternion.SIZE; j++) {
                if (!Arrays.asList(jmaxes).contains(j)) {
                    if (Math.abs(a[i].get(j)) > Math.abs(a[imax].get
                        (jmax))) {
                        imax = i; jmax = j;
                    }
                }
            }
        }
        HashMap<String, Integer> result = new HashMap<String,
            Integer>();
        result.put("i", imax);
        result.put("j", jmax);
        return result;
    }

    public void swapTwoEquations(int i, int step) throws
        IndexException {
        Quaternion ai = a[i];
        a[i] = a[step];
        a[step] = ai;
        double bi = b.get(i);
        b.set(i, b.get(step));
        b.set(step, bi);
    }


}
```

Listing Б.12 — Gauss.java

```
package guseyn.com.methods;

import guseyn.com.exceptions.*;
import guseyn.com.interfaces.NonLinearFunction;
import guseyn.com.libs.QMatrix;
import guseyn.com.libs.Quaternion;
import guseyn.com.libs.Vector;

```

```java
import java.util.Arrays;
import java.util.HashMap;

public class Newton {

    private Quaternion approximation;
    private Quaternion exactValue;
    private NonLinearFunction<Quaternion, Quaternion[]> f;
    private double e;

    private Quaternion[] solution;
    private Quaternion mainDiscrepancy;
    private double[] xi = new double[] {1, 0.5, 0.25,  0.125,
       0.0625,
            0.03125, 0.015625, 0.0078125, 0.00390625, 0.001953125,
            0.000976563, 0.000488281, 0.000244141, 0.00012207,
            0.000061035, 0.000030518, 0.000015259, 0.000007629,
            0.000003815, 0.000001907, 0.000000954, 0.000000477,
            0.000000238, 0.000000119, 0.00000006, 0.00000003,
            0.000000015, 0.0000000075, 0.00000000375,
               0.000000001875,
            0.0000000009375};
    private boolean isNextInvoked = false;

    public Newton(Quaternion approximation, Quaternion exactValue,
                  NonLinearFunction<Quaternion, Quaternion[]> f,
                     double e) {
        this.approximation = approximation;
        this.exactValue = exactValue;
        this.f = f;
        this.e = e;
    }

    public HashMap<String, Object> solve() throws ZeroException,
       MultiplySideException,
            SizeException, IndexException,
               NotResolvableNewtonMethodException {
        HashMap<String, Object> result = new HashMap<String, Object
           >();
        int count = 1;
        while (!isExitCondition()) {
            getNextApproximation(count);
            count++;
        }
        result.put("solution", solution);
        result.put("approximation", approximation);
        return result;
    }

```

```java
    private void getNextApproximation(int count) throws SizeException,
            MultiplySideException, ZeroException, NotResolvableNewtonMethodException, IndexException {

        System.out.println();
        System.out.println("################################################################");
        System.out.println(count);
        Quaternion[] diffs = getDiscrepancyDiffs();
        System.out.println("diffs: \n" + Arrays.toString(diffs) + "\n");
        System.out.println("determinant: \n" + QMatrix.getDeterminant(diffs) + "\n");
        Quaternion rightSideOfSLE = mainDiscrepancy.multiplyWithScalar(-1.0);
        this.isNextInvoked = true;

        double mainDiscrepancyValue = getDiscrepancyValue(mainDiscrepancy);
        boolean status = false;

        Quaternion delta = new Gauss(diffs, rightSideOfSLE).solve();
        System.out.println("real b: \n" + rightSideOfSLE + "\n");
        System.out.println("Gauss b: \n" + Gauss.checkB(diffs, delta) + "\n");

        for (double x: xi) {
            System.out.println("================================================================");
            System.out.println("xi: " + x);
            Quaternion newApproximation = new Quaternion(approximation.plus(delta.multiplyWithScalar(x)));
            System.out.println("new approximation: " + newApproximation);
            Quaternion[] newSolution = f.apply(newApproximation);
            Quaternion newDiscrepancy = new Quaternion(getDiscrepancy(newSolution));
            double newDiscrepancyValue = getDiscrepancyValue(newDiscrepancy);
            System.out.println("new discrepancyValue: " + newDiscrepancyValue + "\n");
            if (newDiscrepancyValue < mainDiscrepancyValue) {
                this.approximation = new Quaternion(newApproximation);
                this.solution = newSolution;
                this.mainDiscrepancy = new Quaternion(newDiscrepancy);
```

```java
                status = true;
                break;
            }
        }

        System.out.println("approximation: \n" + approximation);
        System.out.println("discrepancy: \n" + mainDiscrepancy);
        System.out.println("discrepancyValue: \n" + getDiscrepancyValue(mainDiscrepancy) + "\n");

        if(!status) {
            throw new NotResolvableNewtonMethodException("not resolvable");
        }
    }

    private Quaternion[] getDiscrepancyDiffs() throws ZeroException, MultiplySideException, SizeException, IndexException {
        double delta = 0.00000001;
        double reverseDelta = 1.0 / delta;

        Quaternion[] f1 = f.apply(new Quaternion(approximation.getScalar() + delta,
                approximation.getVector()));
        Quaternion[] f2 = f.apply(new Quaternion(approximation.getScalar(),
                new Vector(approximation.getICoefficient() + delta,
                        approximation.getJCoefficient(),
                        approximation.getKCoefficient())));
        Quaternion[] f3 = f.apply(new Quaternion(approximation.getScalar(),
                new Vector(approximation.getICoefficient(),
                        approximation.getJCoefficient() + delta,
                        approximation.getKCoefficient())));
        Quaternion[] f4 = f.apply(new Quaternion(approximation.getScalar(),
                new Vector(approximation.getICoefficient(),
                        approximation.getJCoefficient(),
                        approximation.getKCoefficient() + delta)));

        return new Quaternion[] {
                getDiscrepancy(f1).minus(mainDiscrepancy).multiplyWithScalar(reverseDelta),
                getDiscrepancy(f2).minus(mainDiscrepancy).multiplyWithScalar(reverseDelta),
                getDiscrepancy(f3).minus(mainDiscrepancy).multiplyWithScalar(reverseDelta),
                getDiscrepancy(f4).minus(mainDiscrepancy).multiplyWithScalar(reverseDelta)
        };
```

```java

    }

    private Quaternion getDiscrepancy(Quaternion[] solution) throws
        ZeroException, MultiplySideException, SizeException {
        return solution[solution.length - 1].minus(exactValue);
    }

    private double getDiscrepancyValue(Quaternion discrepancy) {
        return Math.abs(discrepancy.getScalar())
                + Math.abs(discrepancy.getICoefficient())
                + Math.abs(discrepancy.getJCoefficient())
                + Math.abs(discrepancy.getKCoefficient());
    }

    private boolean isExitCondition() throws ZeroException,
        MultiplySideException, SizeException, IndexException {
        if (!isNextInvoked) {
            this.solution = f.apply(approximation);
            this.mainDiscrepancy = getDiscrepancy(solution);
        }
        return getDiscrepancyValue(mainDiscrepancy) < e;
    }


}
```

Listing Б.13 — QualityFunctional.java

```java
package guseyn.com.methods;


import guseyn.com.exceptions.IndexException;
import guseyn.com.libs.Quaternion;

public class QualityFunctional {

    Quaternion[] values;
    double step;
    Double[] alpha;

    public QualityFunctional(Quaternion[] values, double step,
        Double[] alpha) {
        this.values = values;
        this.step = step;
        this.alpha = alpha;
    }

    public double get() throws IndexException {
        double result = f(0) + f(values.length - 1);
```

```java
        for (int i = 0; i < values.length; i++) {
            if (i != values.length - 1) {
                if (i % 2 == 0) {
                    result += 2 * f(i);
                } else {
                    result += 4 * f(i);
                }
            }
        }
        return result * (step / 3.0);
    }

    private double f(int j) throws IndexException {
        return alpha[0] * Math.pow(values[j].get(1), 2)
                + alpha[1] * Math.pow(values[j].get(2), 2)
                + alpha[2] * Math.pow(values[j].get(3), 2);
    }

}
```

Listing Б.14 — RungeKutta.java

```java
package guseyn.com.methods;

import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;
import guseyn.com.exceptions.ZeroException;
import guseyn.com.interfaces.OrientationFunction;
import guseyn.com.libs.Quaternion;

import java.util.HashMap;

public class RungeKutta {

    protected double leftEdgeOfSegment;
    protected double rightEdgeOfSegment;
    protected double step;

    public RungeKutta() {}

    public RungeKutta(double leftEdgeOfSegment, double
       rightEdgeOfSegment, double step) {
        this.leftEdgeOfSegment = leftEdgeOfSegment;
        this.rightEdgeOfSegment = rightEdgeOfSegment;
        this.step = step;
    }

    public HashMap solve(Quaternion quaternion, OrientationFunction<
       Double, Quaternion, Quaternion> f)
```

```java
            throws ZeroException, MultiplySideException, SizeException {

        if (step == 0) {
            throw new ZeroException("step is 0");
        }

        int quantityOfIntervals =
                (int) ((rightEdgeOfSegment - leftEdgeOfSegment) / step);

        HashMap<String, Quaternion[]> result = new HashMap<String, Quaternion[]>();
        double[] time = new double[quantityOfIntervals];
        Quaternion[] solution = new Quaternion[quantityOfIntervals];

        time[0] = leftEdgeOfSegment;
        solution[0] = quaternion;

        for (int i = 1; i < quantityOfIntervals; i++) {
            HashMap nextInRKMethod = getNextInRungeKutta(time[i - 1],
                    solution[i - 1], f, step);
            time[i] = (Double) nextInRKMethod.get("time");
            solution[i] = (Quaternion) nextInRKMethod.get("quaternion");
        }

        result.put("quaternion", solution);
        return result;

    }

    protected static HashMap<String, Object> getNextInRungeKutta(Double currentTime,
Quaternion currentQuaternion,OrientationFunction<Double, Quaternion, Quaternion> f,
double step) throws MultiplySideException, SizeException {
        final double HF = 0.5;
        final double ONE_SIXTH = 1.0 / 6;
        double halfOfStep = HF * step;

        HashMap<String, Object> result = new HashMap<String, Object>();

        Quaternion k1 = new Quaternion(f.apply(currentTime, currentQuaternion));
        Quaternion k2 = new Quaternion(f.apply(currentTime + halfOfStep, currentQuaternion
                .plus(k1.multiplyWithScalar(halfOfStep))));
```

```java
        Quaternion k3 = new Quaternion(f.apply(currentTime +
            halfOfStep, currentQuaternion
                .plus(k2.multiplyWithScalar(halfOfStep))));
        Quaternion k4 = new Quaternion(f.apply(currentTime + step,
            currentQuaternion
                .plus(k3.multiplyWithScalar(step))));

        Quaternion delta = new Quaternion((k1
                .plus(k2.multiplyWithScalar(2))
                .plus(k3.multiplyWithScalar(2))
                .plus(k4)
        ).multiplyWithScalar(step * ONE_SIXTH));

        result.put("time", currentTime + step);
        result.put("quaternion", currentQuaternion.plus(delta));

        return result;

    }

}
```

Listing Б.15 — RungeKuttaSpecific.java

```java
package guseyn.com.methods;

import guseyn.com.exceptions.IndexException;
import guseyn.com.exceptions.MultiplySideException;
import guseyn.com.exceptions.SizeException;
import guseyn.com.exceptions.ZeroException;
import guseyn.com.interfaces.OrientationFunction;
import guseyn.com.libs.Quaternion;
import guseyn.com.libs.Vector;

import java.util.HashMap;

public class RungeKuttaSpecific extends RungeKutta {

    protected Quaternion lambdaQuaternion;
    protected Double[] alpha;

    public RungeKuttaSpecific(double leftEdgeOfSegment,
                              double rightEdgeOfSegment,
                              double step,
                              Quaternion lambdaQuaternion,
                              Double[] alpha) {
        super(leftEdgeOfSegment, rightEdgeOfSegment, step);
        this.lambdaQuaternion = new Quaternion(lambdaQuaternion);
        this.alpha = alpha;
    }
```

```java

    public HashMap solve(Quaternion psiQuaternion) throws
        ZeroException,
            MultiplySideException,
            SizeException, IndexException {

        if (step == 0) {
            throw new ZeroException("step is 0");
        }

        int quantityOfIntervals =
                (int) ((rightEdgeOfSegment - leftEdgeOfSegment) /
                    step);

        HashMap<String, Quaternion[]> result = new HashMap<String,
            Quaternion[]>();
        Quaternion[] lambdaResult = new Quaternion[
            quantityOfIntervals];
        Quaternion[] psiResult = new Quaternion[quantityOfIntervals
            ];
        Quaternion[] omegaResult = new Quaternion[
            quantityOfIntervals];
        double[] time = new double[quantityOfIntervals];

        time[0] = leftEdgeOfSegment;
        lambdaResult[0] = new Quaternion(lambdaQuaternion);
        psiResult[0] = new Quaternion(psiQuaternion);
        omegaResult[0] = new Quaternion(getOmegaOptimal(lambdaResult
            [0], psiResult[0], alpha));

        for (int i = 1; i < quantityOfIntervals; i++) {
            HashMap next = getNextInSpecificRungeKutta(time[i - 1],
                    lambdaResult[i - 1], psiResult[i - 1],
                        omegaResult[i - 1]);
            time[i] = (Double) next.get("time");
            lambdaResult[i] = new Quaternion((Quaternion) next.get("
                lambda"));
            psiResult[i] = new Quaternion((Quaternion) next.get("psi
                "));
            omegaResult[i] = new Quaternion((Quaternion) next.get("
                omega"));
        }

        result.put("lambda", lambdaResult);
        result.put("psi", psiResult);
        result.put("omega", omegaResult);
        return result;
    }

    protected HashMap<String, Object> getNextInSpecificRungeKutta(
```

```
Double time, final Quaternion lambdaQuaternion,
Quaternion psiQuaternion, final Quaternion omegaOptimalQuaternion)
throws MultiplySideException, SizeException, IndexException {

        HashMap<String, Object> result = new HashMap<String, Object
            >();

        Quaternion newLambdaQuaternion = new Quaternion((Quaternion)
             getNextInRungeKutta(time, lambdaQuaternion,
                  new OrientationFunction<Double, Quaternion,
                      Quaternion>() {
                        @Override
                        public Quaternion apply(Double time, Quaternion
                            lambda) throws SizeException,
                            MultiplySideException {
                              return lambda
                                      .multiplyWithQuaternion(
                                          omegaOptimalQuaternion, "right")
                                      .multiplyWithScalar(0.5);
                        }
                  }, step).get("quaternion"));

        Quaternion newPsiQuaternion = new Quaternion((Quaternion)
            getNextInRungeKutta(time, psiQuaternion,
                  new OrientationFunction<Double, Quaternion,
                      Quaternion>() {
                        @Override
                        public Quaternion apply(Double time, Quaternion
                            psi) throws SizeException,
                            MultiplySideException {
                              return psi
                                      .multiplyWithQuaternion(
                                          omegaOptimalQuaternion, "right")
                                      .multiplyWithScalar(0.5);
                        }
                  }, step).get("quaternion"));

        double newTime = time + step;

        result.put("time", newTime);
        result.put("lambda", newLambdaQuaternion);
        result.put("psi", newPsiQuaternion);
        result.put("omega", new Quaternion(getOmegaOptimal(
            newLambdaQuaternion, newPsiQuaternion, alpha)));

        return result;
    }

```

```java
    public static Quaternion getOmegaOptimal(Quaternion
        lambdaQuaternion, Quaternion psiQuaternion, Double[] alpha)
        throws IndexException {
        double psi0 = psiQuaternion.get(0);
        double psi1 = psiQuaternion.get(1);
        double psi2 = psiQuaternion.get(2);
        double psi3 = psiQuaternion.get(3);
        double lambda0 = lambdaQuaternion.get(0);
        double lambda1 = lambdaQuaternion.get(1);
        double lambda2 = lambdaQuaternion.get(2);
        double lambda3 = lambdaQuaternion.get(3);

        double p1 = (-psi0 * lambda1 + psi1 * lambda0 + psi2 *
            lambda3 - psi3 * lambda2) / (4 * alpha[0]);
        double p2 = (-psi0 * lambda2 - psi1 * lambda3 + psi2 *
            lambda0 + psi3 * lambda1) / (4 * alpha[1]);
        double p3 = (-psi0 * lambda3 + psi1 * lambda2 - psi2 *
            lambda1 + psi3 * lambda0) / (4 * alpha[2]);

        return new Quaternion(0, new Vector(p1, p2, p3));
    }

}
```

Listing Б.16 — Main.java

```java
package guseyn.com.main;

import guseyn.com.exceptions.*;
import guseyn.com.interfaces.NonLinearFunction;
import guseyn.com.libs.Quaternion;
import guseyn.com.methods.Newton;
import guseyn.com.methods.QualityFunctional;
import guseyn.com.methods.RungeKuttaSpecific;

import java.io.IOException;
import java.util.HashMap;

public class Main {

    final static String LAMBDA_PATH = "/home/guseyn/diploma/src/
        guseyn/com/output/plot2d-lambda.r";
    final static String OMEGA_PATH = "/home/guseyn/diploma/src/
        guseyn/com/output/plot2d-omega.r";
    final static String PSI_PATH = "/home/guseyn/diploma/src/guseyn/
        com/output/plot2d-psi.r";
    final static String ANGELS_ALPHA_PATH = "/home/guseyn/diploma/
        src/guseyn/com/output/plot2d-angels-alpha.r";
    final static String ANGELS_BETA_PATH = "/home/guseyn/diploma/src
        /guseyn/com/output/plot2d-angels-beta.r";
```

```java
    final static String ANGELS_GAMMA_PATH = "/home/guseyn/diploma/src/guseyn/com/output/plot2d-angels-gamma.r";

    protected static Quaternion lambda_t0;
    protected static Quaternion lambda_T;
    protected static double e;

    protected static Double leftEdgeOfSegment;
    protected static Double rightEdgeOfSegment;
    protected static Double step;
    protected static Double[] alpha;
    protected static Quaternion approximation;

    private static void graph(Quaternion[] solution, Quaternion[] omega, Quaternion[] psi) throws IndexException, IOException {
        GraphString.$$(solution, LAMBDA_PATH, step, true, 500, -1, 1,
                "text(95, 0, expression(lambda[0]));"
                        + "text(180, -0.70, expression(lambda[1]));"
                        + "text(25, 0.75, expression(lambda[3]));"
                        + "text(75, -0.55, expression(lambda[4]))");
        GraphString.$$(omega, OMEGA_PATH, step, false, 500, -0.02, 0.02,
                "text(10, 0.01, expression(omega[1]));"
                        + "text(10, 0.0, expression(omega[2]));"
                        + "text(10, -0.012, expression(omega[3]))");
        GraphString.$$(psi, PSI_PATH, step, true, 500, -180000, 180000,
                "text(95, 0, expression(psi[0]));"
                        + "text(75, 111000, expression(psi[3]));"
                        + "text(75, -115000, expression(psi[2]));"
                        + "text(27, 0, expression(psi[1]))");
        double[] alpha = new double[solution.length];
        double[] beta = new double[solution.length];
        double[] gamma = new double[solution.length];
        double[] x = new double[solution.length];
        for (int i = 0; i < solution.length; i++) {
            Double[] angels = Quaternion.getEulerAngels(solution[i]);
            alpha[i] = angels[0];
            beta[i] = angels[1];
            gamma[i] = angels[2];
            x[i] = i * step;
        }
        GraphString.$(x, alpha, "t", "alpha", "t", "  ", 200, true, ANGELS_ALPHA_PATH);
        GraphString.$(x, beta, "t", "beta", "t", "  ", 200, true, ANGELS_BETA_PATH);
        GraphString.$(x, gamma, "t", "gamma", "t", "  ", 200, true, ANGELS_GAMMA_PATH);
```

```java
    }

    public static void solve() throws MultiplySideException,
        ZeroException, IndexException, SizeException,
        NotResolvableNewtonMethodException, IOException {

        final RungeKuttaSpecific rungeKuttaSpecific = new
            RungeKuttaSpecific(leftEdgeOfSegment,
                rightEdgeOfSegment, step, lambda_t0, alpha);

        NonLinearFunction<Quaternion, Quaternion[]> RK = new
            NonLinearFunction<Quaternion, Quaternion[]>() {
            @Override
            public Quaternion[] apply(Quaternion psi) throws
                ZeroException,
                    MultiplySideException,
                    SizeException, IndexException {
                return (Quaternion[]) rungeKuttaSpecific.solve(psi).
                    get("lambda");
            }
        };

        HashMap result = new Newton(approximation, lambda_T, RK, e).
            solve();
        Quaternion[] solution = (Quaternion[]) result.get("solution"
            );
        Quaternion approximation = (Quaternion) result.get("
            approximation");
        Quaternion[] omega = (Quaternion[]) rungeKuttaSpecific.solve
            (approximation).get("omega");
        Quaternion[] psi = (Quaternion[]) rungeKuttaSpecific.solve(
            approximation).get("psi");
        double quality = new QualityFunctional(omega, step, alpha).
            get();
        System.out.println("omega0:\n" + omega[0]);
        System.out.println("psi0:\n" + psi[0]);
        System.out.println("quality: \n" + quality);

        graph(solution, omega, psi);
    }

    public static void main(String[] args) throws Exception {

    }

}
```