Московский авиационный институт

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)  
–––––––––––––––  
Аэрокосмический факультет   
––––––––––––––––  
кафедра 611Б «системный анализ и управление»

Проектная работа

по дисциплине: «Моделирование систем»

на тему: «Моделирование движения ИСЗ»

выполнили:

студенты группы 60-307Б

Карпухина А.В.

Савенков В.Р.

Щербатых К.С.

Москва 2021г

Оглавление

[**Введение** 3](#_Toc74101413)

[**Постановка задачи** 3](#_Toc74101414)

[**Необходимая теоретическая база для выполнения задачи** 3](#_Toc74101415)

[**Способ решения** 6](#_Toc74101416)

[**Серверная часть** 6](#_Toc74101417)

[**Клиентская часть** 8](#_Toc74101418)

[**Итог** 10](#_Toc74101419)

[**Листинг** 10](#_Toc74101420)

# **Введение**

По дисциплине “Моделирование систем” было предложено выполнить проектную работу в одном из следующих направлений на выбор:

1. Мобильная разработка
2. САПР
3. Gaming-проект
4. Веб-разработка
5. back-end разработка
6. ERP-система
7. Desktop-приложение

Нашей командой было принято решение выбрать из представленных направлений Desktop-приложение.

# **Постановка задачи**

Прогнозирование движения КЛА является одной из целевых задач науки, которая имеет такие названия как космическая баллистика, механика космического полёта, прикладная небесная механика, космодинамика, теория движения искусственных небесных тел, космонавтика и т.д.

В следствии этого было решено следующее:

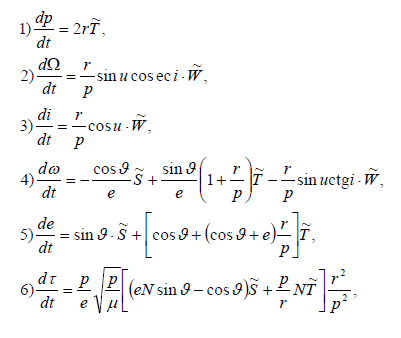
Создать проект, представляющий собой клиент-серверное приложение, целью которого является моделирование движений спутников ИСЗ (искусственного спутника Земли)

# **Необходимая теоретическая база для выполнения задачи**

Для построения модели возмущенного движения используется метод оскулирующих элементов.

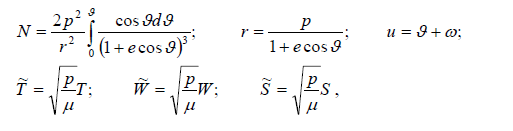
Метод разработан Лагранжем и получил широкое распространение при исследовании возмущенного движения. Сущность метода заключается в том, что возмущенную (истинную) траекторию КА рассматривают как состоящую из последовательности невозмущенных траекторий с разными параметрами для каждого текущего момента времени. В итоге траектория возмущенного движения в каждый момент времени соприкасается с траекторией невозмущенного движения для этого же момента времени и представляет собой огибающую семейства невозмущенных траекторий движения.

В результате имеем следующую систему уравнений

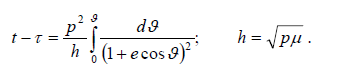


(1)

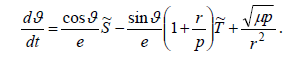
Где



а истинная аномалия θ связана со временем t формулой

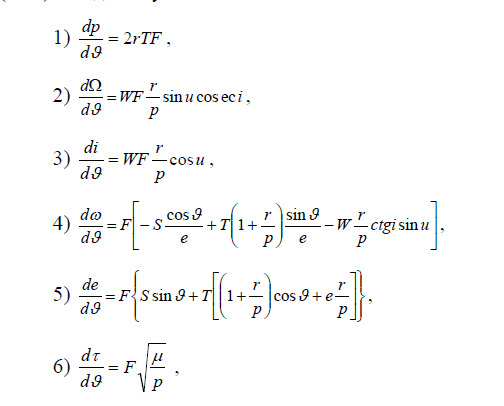


На величины возмущающих ускорений *S,T,W*, входящих в систему уравнений не наложено никаких ограничений. Если возмущающие ускорения *S,T,W* не зависят явно от времени *t*, то правые части уравнений будут функциями только от истинной аномалии и элементов орбиты. Поэтому в системе можно исключить время, приняв за независимую переменную θ . Можно показать, что связь между θ и t определяется соотношением:



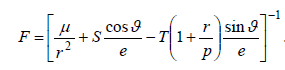
(2)

Каждое из уравнений системы (1) разделим почленно на уравнение (2). Тогда получим:



(3)

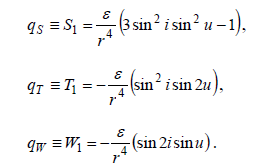
Где



(4)

Для решения системы дифференциальных уравнений (3) используется метод Эйлера. Приближенное решение в узлах xi, которое обозначим через yi, определяется по формуле:

Возмущающие ускорения, вызываемые нецентральностью поля тяготения Земли, определяеются как:



# **Способ решения**

Способ реализации включал в себя создание клиентской и серверной части приложения.

## **Серверная часть**

Главная задача серверной части заключалась в вычислении нового положения зарегистрированных ИСЗ.

Сервер написан на языке Python с использованием веб фреймворка Flask. Как уже было сказано раннее на сервере происходят основные вычисления, связанные с прогнозированием КЛА в разные моменты времени. Общение с клиентом происходит по HTTP протоколу, данные отдаются в формате JSON через GET запрос с клиента.

Основные используемые библиотеки для реализации серверной части:

* Библиотека и парсинга и сериализации JSON – json. С её помощью стало удобным хранение и передача данных в формате JSON. Для записи данных был создан метод, в котором использовалась функция dumps() используемого модуля, позволяющая сериализовать объект Python  в строку формата JSON.
* Используемый фреймворк – flask. Для создания flask-приложения был создан обязательный экземпляр: конструктору Flask назначаем аргумент \_\_name\_\_. Также создадим маршрут для привязки URL к функции представления, отвечающей за запрос. В используемом нами фреймворке flask, декоратор route мы использовали, чтобы связать URL адрес с функцией. Функция index является здесь обработчиком корневого URL в приложении.
* Также использовали модуль sqlite3.

С использованием этого модуля мы создавали базу данных SQLite, без скачивания дополнительных инструментов. Модуль также позволил нам вести запись в локальное хранилище с целью повторного использования и анализа движения.

**SQLite** – автономный, работающий без сервера транзакционный механизм базы данных SQL. Далее были реализованы две функции для создания таблицы и её заполнение данными (координатами и соотвествующими данными об оскулирующих элементах) и функция создания подключения.

Результирующие таблицы – Таблица1,2.

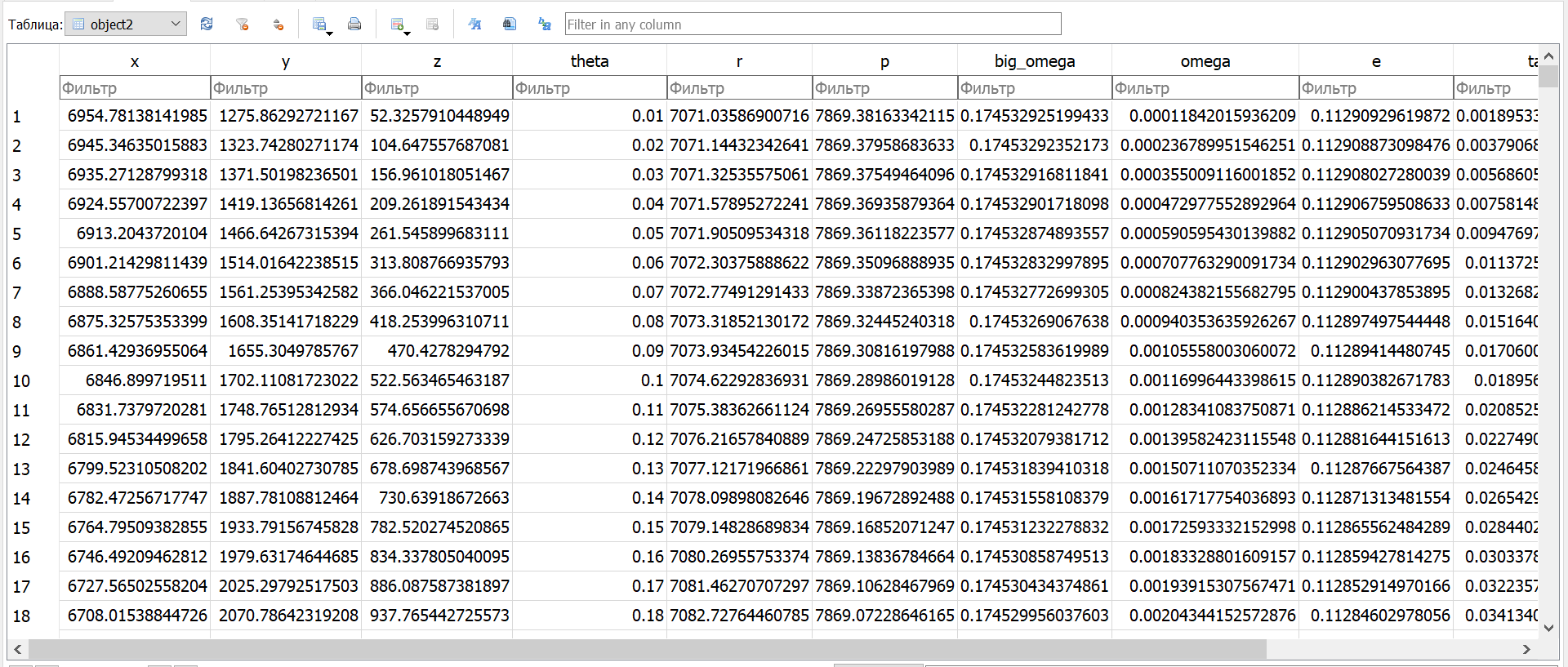


Таблица 1. – Таблица для object2 с его параметрами.

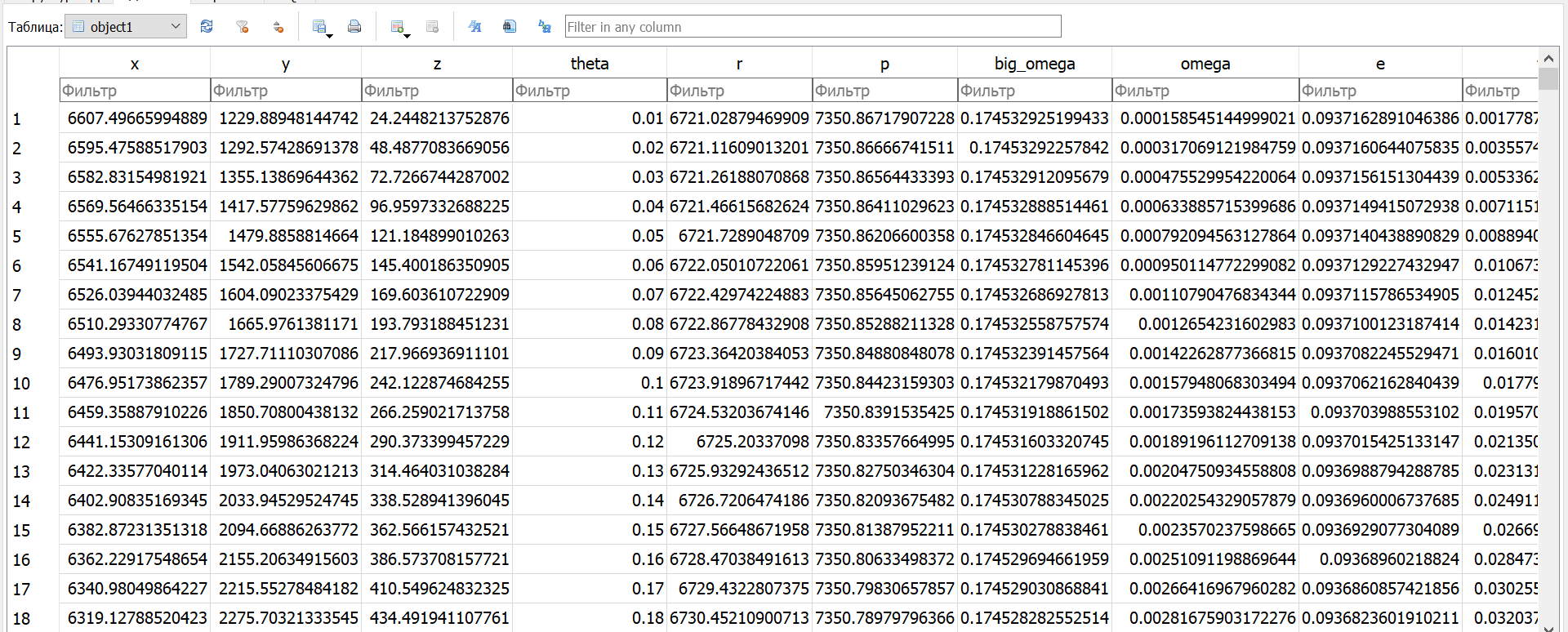


Таблица 2. - Таблица для object1 с его параметрами.

Также на сервере был реализована передача по HTTP протоколу с соответствующими обработчиками GET запросов, предназначенных для взаимодействия с клиентом (API).

## **Клиентская часть**

Главная задача клиентской части заключалась в запечатлении положения ИСЗ в текущий момент времени, выводе характеристик, а также задании новых ИСЗ.

Клиентская часть написана на языке С++ и фреймворке QT. Клиент отображает положение ИСЗ в текущий момент времени в 3D пространстве и в проекциях, также выводит различные характеристики, касающиеся ИСЗ.

Компьютерная графика реализована с помощью программного обеспечения OpenGL.

В работе с OpenGL рассмотрим использованные виртуальные функции:

* initializeGL()
* resizeGL()
* paintGL()

Через эти функции будет осуществляться все взаимодействие с OpenGL – инициализация, настройка, непосредственно отрисовка графических примитивов, фигур и тд.

* Функция initializeGL() нужна для того, чтобы произвести в ней всю необходимую инициализацию, настроить параметры отображения, изменить режим работы OpenGL.
* Функция resizeGL() вызывается каждый раз при изменении размеров окна.
* Функция paintGL() вызывается каждый раз после вызова функции resizeGL() и запускает перерисовку сцены. Кроме этого случая, paintGL() вызывается каждый раз после вызова функции updateGL().

Отдельное внимание стоит уделить отрисовке 3D объектов с использованием OpenGL. Отрисовка производилась с использованием графической примитивы – треугольника. Процесс отрисовки сферы производилась путем перебора всех точек на её поверхности. Точки использовались в роли связующего для множества граней треугольника (Рисунок 1).

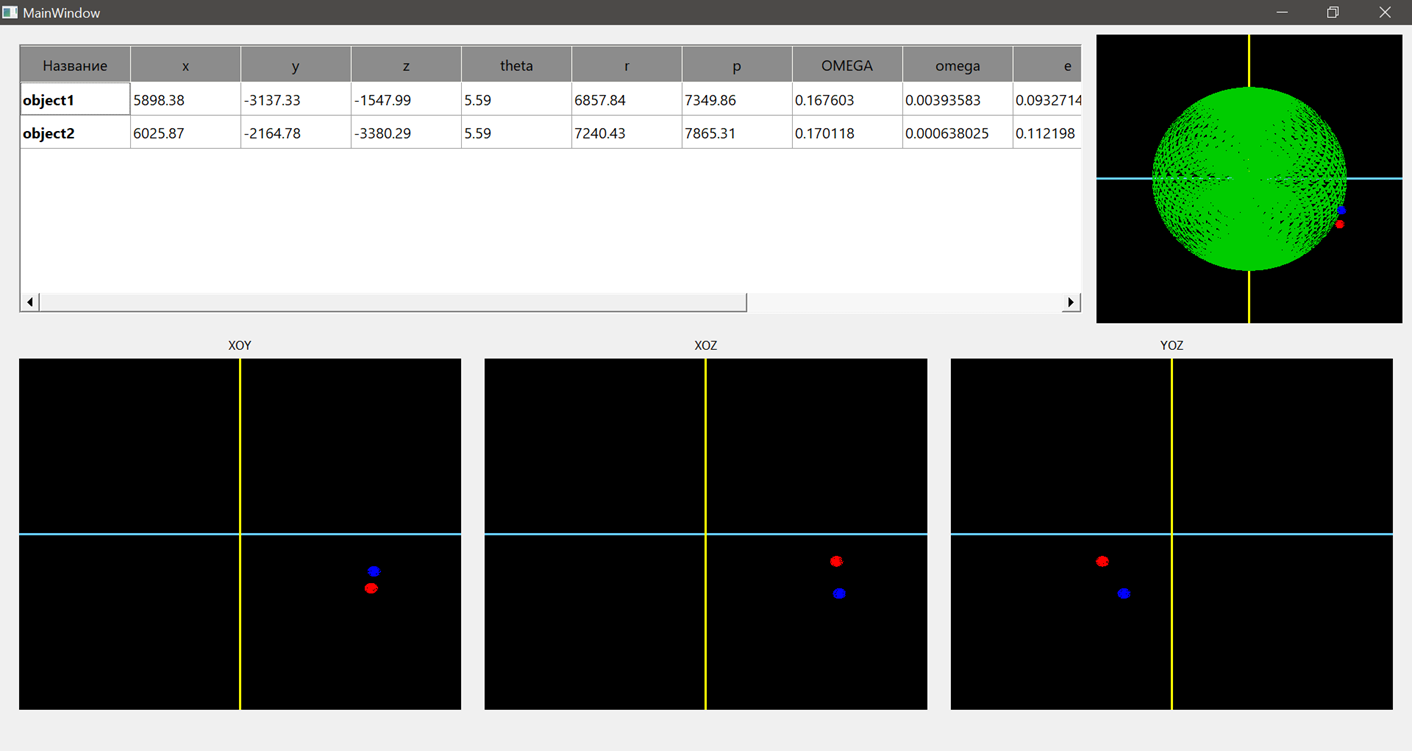


Рисунок 1 – Интерфейс.

Отдельно полученные параметры ИСЗ:

Object1:

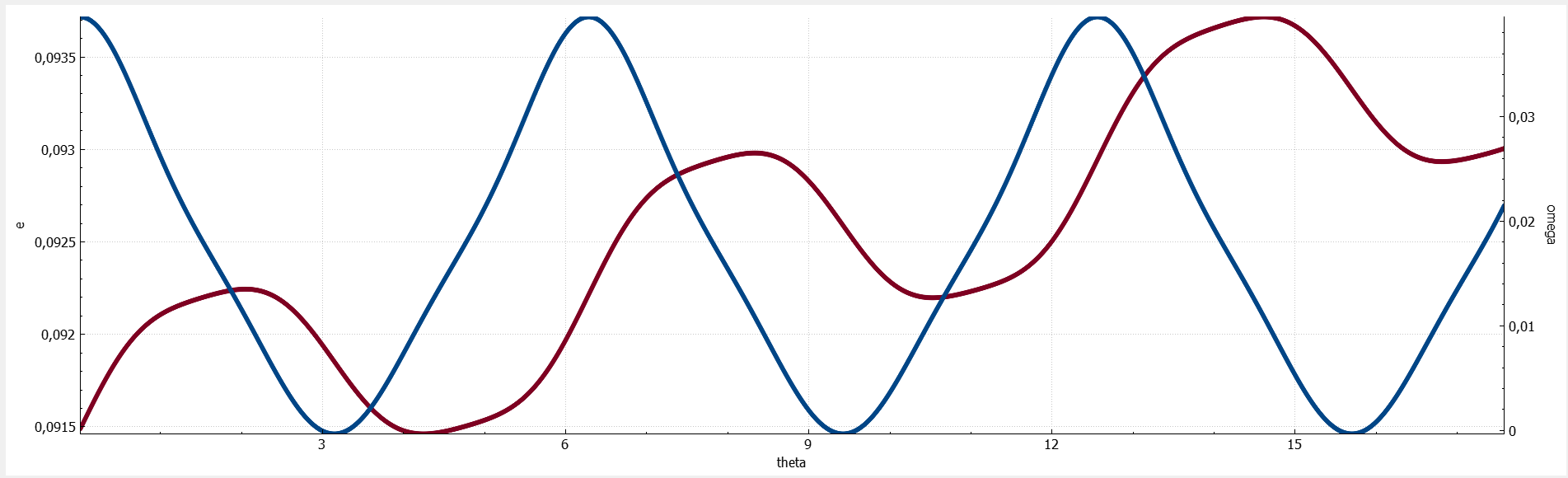


График 1. - Зависимости эксцентриситета и аргумента перицентра от тетты.

Object2:

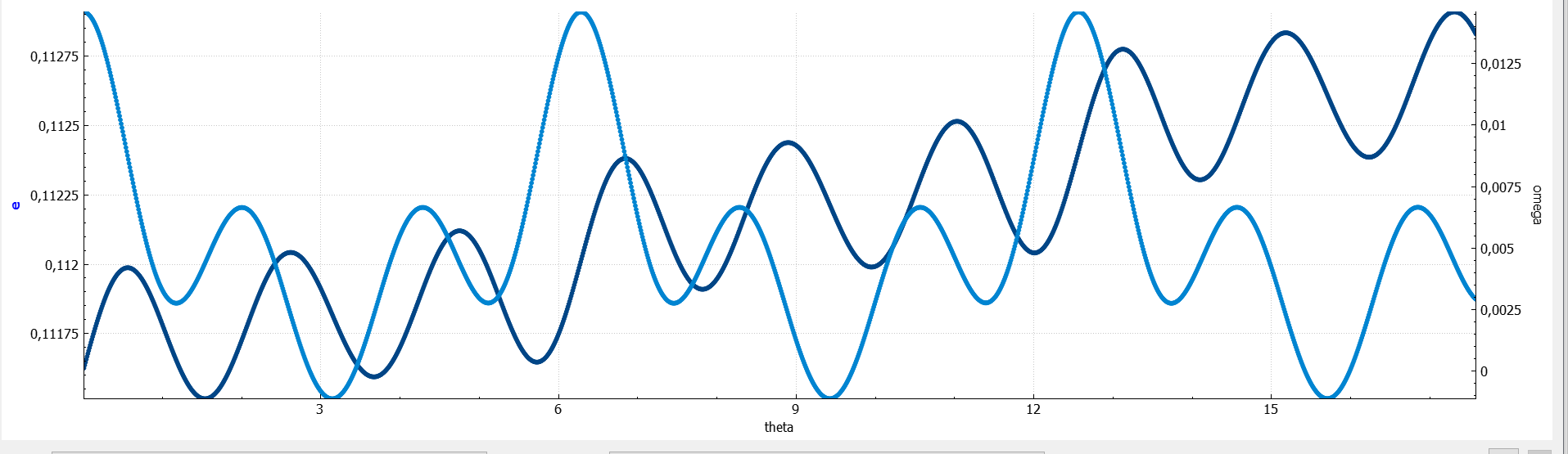


График 2. - Зависимости эксцентриситета и аргумента перицентра от тетты.

# **Итог**

В итоге мы:

* Познакомились с фреймворком Flask
* Поработали с библиотеками JSON, SQLlite3
* Научились разрабатывать трехмерные модели с использованием графической примитивы и библиотеки OpenGL
* Углубили знания в теоретическом материале по динамике летательных аппаратов.

# **Листинг**

Клиент

1. #include "mainscene.h"
2. #include <QApplication>
3. #include <mainwindow.h>
5. int main(int argc, char **\***argv[])
6. {
7. QApplication a(argc, argv);
8. MainWindow w;
9. w.show();
10. **//**    MainScene scene;
11. **//**    scene.resize(700, 700);
12. **//**    scene.show();
14. **return** a.**exec**();
15. }
16. #include "mainscene.h"
18. MainScene::MainScene()
19. {
20. \_proectionType **=** Default;
21. timer **=** new QTimer;
22. connect(timer, &QTimer::timeout, this, &MainScene::TimerAlarm);
23. timer**-**>start(rendering\_delay\_ms);
24. }
26. MainScene::MainScene(ProectionType type):\_proectionType(type)
27. {
29. }
31. void MainScene::mousePressEvent(QMouseEvent **\***event)
32. {
33. pressPosition **=** event**-**>pos();
34. }
36. void MainScene::mouseMoveEvent(QMouseEvent **\***event)
37. {
38. xAxisRotation **+=** (180 **\*** ((GLfloat)event**-**>y() **-** (GLfloat)pressPosition.y())) **/** (currentHeight);
39. yAxisRotation **+=** (180 **\*** ((GLfloat)event**-**>x() **-** (GLfloat)pressPosition.x())) **/** (currentWidth);
41. pressPosition **=** event**-**>pos();
42. updateGL();
43. }
45. void MainScene::initializeGL()
46. {
47. glClearColor(1.0, 1.0, 1.0, 1.0);
48. glClearColor(0.0, 0.0, 0.0, 1.0);
49. glEnableClientState(GL\_VERTEX\_ARRAY);
50. glEnableClientState(GL\_COLOR\_ARRAY);
52. glShadeModel(GL\_FLAT);
53. glEnable(GL\_CULL\_FACE);
54. }
56. void MainScene::resizeGL(int w, int h)
57. {
58. currentHeight **=** h;
59. currentWidth **=** w;
60. glMatrixMode(GL\_PROJECTION);
61. glLoadIdentity();
62. **//**    glOrtho(**-**20.0, 20.0, **-**20.0, 20.0, **-**20.0, 20.0);
63. glOrtho(beg\_x, end\_x, beg\_y, end\_y, beg\_z, end\_z);
64. glViewport(0, 0, (GLint)w, (GLint)h);
65. glMatrixMode(GL\_MODELVIEW);
66. }
68. void MainScene::paintGL()
69. {
70. **//**    glClear(GL\_COLOR\_BUFFER\_BIT);
72. glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);
73. glMatrixMode(GL\_MODELVIEW);
74. glLoadIdentity();
75. glRotatef(yAxisRotation, 0.0, 1.0, 0.0);
76. glRotatef(xAxisRotation, 1.0, 0.0, 0.0);
77. **//**   glClear(GL\_COLOR\_BUFFER\_BIT);
79. double lineWidth **=** 3;
81. **//** ОСЬ Х
82. **//** **==============================================**
83. **//**    glColor3f(158**/**255.0, 164**/**255.0, 254**/**255.0);
84. glColor3f(102**/**255.0, 204**/**255.0, 255**/**255.0);
85. glLineWidth(lineWidth);
86. glBegin(GL\_LINE\_STRIP);
87. glVertex3f(beg\_x, 0, 0);
88. glVertex3f(end\_x, 0, 0);
89. glEnd();
90. **//** **==============================================**
92. **//** ОСЬ Y
93. **//** **==============================================**
94. glColor3f(1, 1, 0);
95. glLineWidth(lineWidth);
96. glBegin(GL\_LINE\_STRIP);
97. glVertex3f(0, beg\_y, 0);
98. glVertex3f(0, end\_y, 0);
99. glEnd();
100. **//** **==============================================**
102. **//** ОСЬ Z
103. **//** **==============================================**
104. glColor3f(0, 1, 1);
105. glLineWidth(lineWidth);
106. glBegin(GL\_LINE\_STRIP);
107. glVertex3f(0, 0, beg\_z);
108. glVertex3f(0, 0, end\_z);
109. glEnd();
110. **//** **==============================================**
112. **if** (\_proectionType **==** Default)
113. {
114. paintSphere(6371,0,0,0, 0, 204**/**255.0, 0);
116. auto fo **=** http.get\_space\_objects();
117. **for** (int i**=**0; i<fo.size(); i**++**)
118. {
119. **if** (i**%**2 **==** 0)
120. paintSphere(300,fo[i].x , fo[i].y, fo[i].z, 1, 0, 0);
121. **else**
122. paintSphere(300,fo[i].x , fo[i].y, fo[i].z, 0, 0, 1);
123. }
124. }
125. **else**
126. {
127. auto fo **=** http.get\_space\_objects();
128. **for** (int i**=**0; i<fo.size(); i**++**)
129. {
130. switch (\_proectionType) {
131. case XOY:
132. {
133. **if** (i **%** 2 **==** 0)
134. paintSphere(300,fo[i].x , fo[i].y, 0, 1, 0, 0);
135. **else**
136. paintSphere(300,fo[i].x , fo[i].y, 0, 0, 0, 1);
137. **break**;
138. }
140. case XOZ:
141. {
142. **if** (i **%** 2 **==** 0)
143. paintSphere(300,fo[i].x , fo[i].z, 0, 1, 0, 0);
144. **else**
145. paintSphere(300,fo[i].x , fo[i].z, 0, 0, 0, 1);
146. **break**;
147. }
149. case YOZ:
150. {
151. **if** (i **%** 2 **==** 0)
152. paintSphere(300,fo[i].y , fo[i].z, 0, 1, 0, 0);
153. **else**
154. paintSphere(300,fo[i].y , fo[i].z, 0, 0, 0, 1);
155. **break**;
156. }
158. }
160. }
161. }



166. }
168. void MainScene::paintSphere(double radius, double X0, double Y0 ,double Z0, float red, float green, float blue)
169. {
170. glColor3f(red, green, blue);
172. float x,y,z;
173. float X**=-**M\_PI,Y**=**0;
174. float Z**=**0;
175. float step **=** 0.1f;
176. glBegin(GL\_TRIANGLE\_STRIP);
178. **while**(X<M\_PI)
179. {
180. **while**(Y<2**\***M\_PI)
181. {
182. x**=**radius**\***cos(X)**\***cos(Y);
183. y**=**radius**\***cos(X)**\***sin(Y);
184. z**=**radius**\***sin(X);
185. glVertex3f(X0**+**x,Y0**+**y,Z0**+**z);
187. x**=**radius**\***cos(X)**\***cos(Y);
188. y**=**radius**\***cos(X**+**step)**\***sin(Y);
189. z**=**radius**\***sin(X);
190. glVertex3f(X0**+**x,Y0**+**y,Z0**+**z);
192. x**=**radius**\***cos(X**+**step)**\***cos(Y);
193. y**=**radius**\***cos(X)**\***sin(Y);
194. z**=**radius**\***sin(X**+**step);
195. glVertex3f(X0**+**x,Y0**+**y,Z0**+**z);
196. Y**+=**step;
197. }
198. Y**=**0;
199. X**+=**step;
200. }
201. glEnd();
202. }
204. void MainScene::GenerateTextures()
205. {
207. }
209. void MainScene::TimerAlarm()
210. {
211. http.send\_request(url);

214. paintGL();
215. updateGL();
216. }
218. void MainScene::SetSpaceObjectList(QList<SpaceObject> list)
219. {
220. http.set\_space\_objects(list);
221. paintGL();
222. updateGL();
223. }
224. #include "mainwindow.h"
225. #include "ui\_mainwindow.h"
227. MainWindow::MainWindow(QWidget **\***parent) :
228. QMainWindow(parent),
229. ui(new Ui::MainWindow)
230. {
231. ui**-**>setupUi(this);
232. InitTable();
233. MainScene **\***scene **=** new MainScene;
234. MainScene **\***xoy **=** new MainScene(XOY);
235. MainScene **\***xoz **=** new MainScene(XOZ);
236. MainScene **\***yoz **=** new MainScene(YOZ);
238. scene**-**>setMinimumWidth(300);
240. **//**SetSpaceObjectList
241. connect(&scene**-**>http, &HTTP::SpaceObjectListUpdated, this, &MainWindow::RenderSpaceObjectTable);
243. connect(&scene**-**>http, &HTTP::SpaceObjectListUpdated, xoy, &MainScene::SetSpaceObjectList);
244. connect(&scene**-**>http, &HTTP::SpaceObjectListUpdated, xoz, &MainScene::SetSpaceObjectList);
245. connect(&scene**-**>http, &HTTP::SpaceObjectListUpdated, yoz, &MainScene::SetSpaceObjectList);
247. ui**-**>OpenglLayout**-**>addWidget(scene);
249. ui**-**>XoyOpenGlWidgetLayout**-**>addWidget(xoy);
250. ui**-**>XozOpenGlWidgetLayout**-**>addWidget(xoz);
251. ui**-**>YozOpenGlWidgetLayout**-**>addWidget(yoz);
252. }
254. MainWindow::~MainWindow()
255. {
256. delete ui;
257. }
259. void MainWindow::InitTable()
260. {
261. QStringList horizontalHeaders;
262. ui**-**>SpaceObjectTable**-**>setRowCount(0);
263. ui**-**>SpaceObjectTable**-**>setColumnCount(0);
265. ui**-**>SpaceObjectTable**-**>setColumnCount(16**-**3);
267. horizontalHeaders << "Название";
268. horizontalHeaders << "x";
269. horizontalHeaders << "y";
270. horizontalHeaders << "z";
271. horizontalHeaders << "theta";
272. **//**    horizontalHeaders << "h\_p";
273. **//**    horizontalHeaders << "h\_a";
274. horizontalHeaders << "r";
275. horizontalHeaders << "p";
276. horizontalHeaders << "OMEGA";
277. horizontalHeaders << "omega";
278. horizontalHeaders << "e";
279. **//**    horizontalHeaders << "U";
280. horizontalHeaders << "Tau";
281. horizontalHeaders << "m";
282. horizontalHeaders << "i";
284. ui**-**>SpaceObjectTable**-**>setHorizontalHeaderLabels(horizontalHeaders);
285. ui**-**>SpaceObjectTable**-**>setRowCount(1);
287. **//**    ui**-**>SpaceObjectTable**-**>horizontalHeader()**-**>setSectionResizeMode(QHeaderView::Stretch);
288. }
290. void MainWindow::RenderSpaceObjectTable(QList<SpaceObject> list)
291. {
292. **//** Установить кол**-**во строк
293. ui**-**>SpaceObjectTable**-**>setRowCount(list.size());
295. **//**Жирный шрифт
296. QFont font;
297. font.setBold(true);

300. **//** Заполнение таблицы
301. **for** (int i**=**0; i<list.size(); i**++**)
302. {
303. ui**-**>SpaceObjectTable**-**>setItem(i, 0, new QTableWidgetItem(list[i].name));
304. ui**-**>SpaceObjectTable**-**>item(i,0)**-**>setFont(font);
305. ui**-**>SpaceObjectTable**-**>setItem(i, 1, new QTableWidgetItem(QString::number(list[i].x)));
306. ui**-**>SpaceObjectTable**-**>setItem(i, 2, new QTableWidgetItem(QString::number(list[i].y)));
307. ui**-**>SpaceObjectTable**-**>setItem(i, 3, new QTableWidgetItem(QString::number(list[i].z)));
308. ui**-**>SpaceObjectTable**-**>setItem(i, 4, new QTableWidgetItem(QString::number(list[i].theta)));
309. **//**        ui**-**>SpaceObjectTable**-**>setItem(i, 5, new QTableWidgetItem(QString::number(list[i].h\_p)));
310. **//**        ui**-**>SpaceObjectTable**-**>setItem(i, 6, new QTableWidgetItem(QString::number(list[i].h\_a)));
311. ui**-**>SpaceObjectTable**-**>setItem(i, 5, new QTableWidgetItem(QString::number(list[i].r)));
312. ui**-**>SpaceObjectTable**-**>setItem(i, 6, new QTableWidgetItem(QString::number(list[i].p)));
313. ui**-**>SpaceObjectTable**-**>setItem(i, 7, new QTableWidgetItem(QString::number(list[i].OMEGA)));
314. ui**-**>SpaceObjectTable**-**>setItem(i, 8, new QTableWidgetItem(QString::number(list[i].omega)));
315. ui**-**>SpaceObjectTable**-**>setItem(i, 9, new QTableWidgetItem(QString::number(list[i].e)));
316. **//**        ui**-**>SpaceObjectTable**-**>setItem(i, 12, new QTableWidgetItem(QString::number(list[i].U)));
317. ui**-**>SpaceObjectTable**-**>setItem(i, 10, new QTableWidgetItem(QString::number(list[i].tau)));
318. ui**-**>SpaceObjectTable**-**>setItem(i, 11, new QTableWidgetItem(QString::number(list[i].m)));
319. ui**-**>SpaceObjectTable**-**>setItem(i, 12, new QTableWidgetItem(QString::number(list[i].i)));
320. }
321. }
322. #include "spaceobject.h"
324. SpaceObject::SpaceObject(QObject **\***parent)
325. {
327. }
329. SpaceObject::SpaceObject(QString name,double x, double y, double z,
330. double theta,
331. double h\_p, double h\_a,
332. double r, double p,
333. double OMEGA, double omega,
334. double e, double U,
335. double tau, double m,
336. double i) :name(name),
337. x(x), y(y), z(z),
338. theta(theta),
339. h\_p(h\_p), h\_a(h\_a),
340. r(r), p(p),
341. OMEGA(OMEGA),omega(omega),
342. e(e),U(U),tau(tau),m(m),i(i)
343. {
345. }

Сервер

1. **import** math
2. # from AtmosphericDensityModelCoefficients import \*
3. # from data import \*
4. **from** math **import** cos, sin, atan, tan, radians, fabs, sqrt, pi
5. **import** numpy as np
6. Earth\_gravity\_potential **=** 698603 **\*** 3600 **\*** 3600
7. # Формулы для рассчёта компонентов возмущающего ускорения,
8. # вызванного нецентральность гравитационного поля Земли
9. # ===================================================================================
10. # Epsilon = Earth\_gravity\_potential \* 66.07 \* 1e3
11. S1 **=** **lambda** r, i, u: Epsilon **/** r **\*\*** 4 **\*** (3 **\*** math.sin(i) **\*\*** 2 **\*** math.sin(u) **\*\*** 2 **-** 1)
12. T1 **=** **lambda** r, i, u: **-**Epsilon **/** r **\*\*** 4 **\*** math.sin(i) **\*\*** 2 **\*** math.sin(2 **\*** u)
13. W1 **=** **lambda** r, i, u: **-**Epsilon **/** r **\*\*** 4 **\*** math.sin(2 **\*** i) **\*** math.sin(u)
15. S2 **=** **lambda** sigma\_x, density, V, Vr: **-**sigma\_x **\*** density **\*** V **\*** Vr
16. T2 **=** **lambda** sigma\_x, density, V, Vt: **-**sigma\_x **\*** density **\*** V **\*** Vt
17. W2 **=** 0
19. S21 **=** **lambda** sigma\_x, density, V, Vr: **-**sigma\_x **\*** density **\*** V **\*** Vr
20. T21 **=** **lambda** sigma\_x, density, V, Vt: **-**sigma\_x **\*** density **\*** V **\*** Vt
21. W21 **=** **lambda** sigma\_x, density, V, r, i, u: **-**sigma\_x **\*** density **\*** V **\*** 7.292115855306578e**-**05 **\*** r **\*** sin(i) **\*** sin(u)
22. # ===================================================================================
24. # Правые части диф уравнений метода оскулирующих элементов
25. # =====================================================================================================================
26. R\_p **=** **lambda** r, T, F: 2 **\*** r **\*** T **\*** F
28. R\_OMEGA **=** **lambda** W, F, r, p, u, i: W **\*** F **\*** r **/** p **\*** math.sin(u) **/** math.sin(i)
30. R\_i **=** **lambda** W, F, r, p, u: W **\*** F **\*** r **/** p **\*** math.cos(u)
32. R\_omega **=** **lambda** F, S, theta, e, T, r, p, W, i, u: F **\*** (
33. **-**S **\*** math.cos(theta) **/** e **+** T **\*** (1 **+** r **/** p) **\*** math.sin(theta) **/** e **-** W **\*** r **/** p **\*** (1 **/** math.tan(i)) **\*** math.sin(u))
35. R\_e **=** **lambda** F, S, theta, T, r, p, e: F **\*** (S **\*** math.sin(theta) **+** T **\*** ((1 **+** r **/** p) **\*** math.cos(theta) **+** e **\*** r **/** p))
37. R\_tau **=** **lambda** F, p: F **\*** math.sqrt(Earth\_gravity\_potential **/** p)
39. geT\_F **=** **lambda** r, S, T, theta, e, p: (Earth\_gravity\_potential **/** r **\*\*** 2 **+** S **\*** math.cos(theta) **/** e **-** T **\*** (
40. 1 **+** r **/** p) **\*** math.sin(
41. theta) **/** e) **\*\*** **-**1

44. # =====================================================================================================================
46. # Рунге Куттэ
47. # !!!
48. # НЕ РАБОТАЕТ
49. # !!!
50. # ========================================
51. **def** runge\_p(r, S, T, e, p, theta, d\_theta):
52. k1 **=** R\_p(r**=**r, T**=**T, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p))
53. k2 **=** R\_p(r**=**r, T**=**T, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p **+** 0.5 **\*** d\_theta **\*** k1))
54. k3 **=** R\_p(r**=**r, T**=**T, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p **+** 0.5 **\*** d\_theta **\*** k2))
55. k4 **=** R\_p(r**=**r, T**=**T, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e, p**=**p **+** d\_theta **\*** k3))
56. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

59. **def** runge\_OMEGA(S, T, W, r, e, p, omega, i, theta, d\_theta):
60. k1 **=** R\_OMEGA(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p), r**=**r, p**=**p, u**=**theta **+** omega, i**=**i)
62. k2 **=** R\_OMEGA(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p,
63. u**=**theta **+** 0.5 **\*** d\_theta **+** omega, i**=**i)
65. k3 **=** R\_OMEGA(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p,
66. u**=**theta **+** 0.5 **\*** d\_theta **+** omega, i**=**i)
68. k4 **=** R\_OMEGA(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p, u**=**theta **+** d\_theta **+** omega, i**=**i)
69. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

72. **def** runge\_i(S, T, W, r, e, p, omega, theta, d\_theta):
73. k1 **=** R\_i(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p), r**=**r, p**=**p, u**=**theta **+** omega)
74. k2 **=** R\_i(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p,
75. u**=**theta **+** 0.5 **\*** d\_theta **+** omega)
76. k3 **=** R\_i(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p,
77. u**=**theta **+** 0.5 **\*** d\_theta **+** omega)
78. k4 **=** R\_i(W**=**W, F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e, p**=**p), r**=**r, p**=**p, u**=**theta **+** d\_theta **+** omega)
80. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

83. **def** runge\_omega(S, T, W, r, e, p, omega, theta, d\_theta):
84. k1 **=** R\_omega(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p), S**=**S, theta**=**theta, e**=**e, T**=**T, r**=**r, p**=**p, W**=**W, i**=**i,
85. u**=**theta **+** omega)
87. k2 **=** R\_omega(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), S**=**S, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e,
88. T**=**T, r**=**r, p**=**p, W**=**W, i**=**i,
89. u**=**theta **+** 0.5 **\*** d\_theta **+** omega **+** 0.5 **\*** d\_theta **\*** k1)
91. k3 **=** R\_omega(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), S**=**S, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e,
92. T**=**T, r**=**r, p**=**p, W**=**W, i**=**i,
93. u**=**theta **+** 0.5 **\*** d\_theta **+** omega **+** 0.5 **\*** d\_theta **\*** k2)
95. k4 **=** R\_omega(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e, p**=**p), S**=**S, theta**=**theta **+** d\_theta, e**=**e, T**=**T, r**=**r,
96. p**=**p, W**=**W, i**=**i,
97. u**=**theta **+** d\_theta **+** omega **+** d\_theta **\*** k3)
98. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

101. **def** runge\_e(S, theta, T, r, p, e, d\_theta):
102. k1 **=** R\_e(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p), S**=**S, theta**=**theta, T**=**T, r**=**r, p**=**p, e**=**e)
104. k2 **=** R\_e(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e **+** 0.5 **\*** d\_theta **\*** k1, p**=**p), S**=**S,
105. theta**=**theta **+** 0.5 **\*** d\_theta, T**=**T, r**=**r, p**=**p, e**=**e **+** 0.5 **\*** d\_theta **\*** k1)
107. k3 **=** R\_e(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e **+** 0.5 **\*** d\_theta **\*** k2, p**=**p), S**=**S,
108. theta**=**theta **+** 0.5 **\*** d\_theta, T**=**T, r**=**r, p**=**p, e**=**e **+** 0.5 **\*** d\_theta **\*** k2)
110. k4 **=** R\_e(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e **+** d\_theta **\*** k3, p**=**p), S**=**S,
111. theta**=**theta **+** d\_theta, T**=**T, r**=**r, p**=**p, e**=**e **+** d\_theta **\*** k3)
113. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

116. **def** runge\_tau(r, S, T, e, p, theta, d\_theta):
117. k1 **=** R\_tau(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p), p**=**p)
118. k2 **=** R\_tau(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), p**=**p)
119. k3 **=** R\_tau(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** 0.5 **\*** d\_theta, e**=**e, p**=**p), p**=**p)
120. k4 **=** R\_tau(F**=**geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta **+** d\_theta, e**=**e, p**=**p), p**=**p)
121. **return** d\_theta **/** 6 **\*** (k1 **+** 2 **\*** k2 **+** 2 **\*** k3 **+** k4)

124. # ========================================

127. # Вспомогательные функции
128. # ========================================================
129. get\_a **=** **lambda** ra, rp: 0.5 **\*** (ra **+** rp)
130. get\_e **=** **lambda** ra, rp, a: (ra **-** rp) **/** (2 **\*** a)
131. get\_r **=** **lambda** p, e, theta: p **/** (1 **+** e **\*** math.cos(theta))
132. get\_p **=** **lambda** a, e: a **\*** (1 **-** e **\*\*** 2)
133. get\_Vr **=** **lambda** p, theta, e: sqrt(Earth\_gravity\_potential **/** p) **\*** e **\*** sin(theta)
134. get\_Vt **=** **lambda** p, theta, e: sqrt(Earth\_gravity\_potential **/** p) **\*** (1 **+** e **\*** cos(theta))
135. # ========================================================
137. get\_sigma\_x **=** **lambda** Cxa, Sa, m: Sa **\*** Cxa **/** (2 **\*** m)

140. **def** get\_t\_from\_tau(tau, theta, p, e):
141. f **=** **lambda** angle: 1 **/** (1 **+** e **\*** math.cos(angle)) **\*\*** 2
142. d\_theta **=** 1e**-**2
143. theta1 **=** 0
144. sum **=** 0
145. **while** theta1 < theta:
146. sum **+=** d\_theta **\*** f(angle**=**theta1 **+** 0.5 **\*** d\_theta)
147. theta1 **+=** d\_theta
149. sum **\*=** math.sqrt(p **/** Earth\_gravity\_potential)
150. sum **+=** tau
151. **return** sum

154. # Функция для получеения АГЭСК координат
155. **def** get\_agesk(theta, ra, i, big\_omega, omega):
156. X **=** ra **\*** (cos(omega **+** theta) **\*** cos(big\_omega) **-** sin(theta **+** omega) **\*** sin(big\_omega) **\*** cos(i))
157. Y **=** ra **\*** (cos(omega **+** theta) **\*** sin(big\_omega) **+** sin(theta **+** omega) **\*** cos(big\_omega) **\*** cos(i))
158. Z **=** ra **\*** sin(omega **+** theta) **\*** sin(i)
159. **return** X, Y, Z

162. # Функция для получения матрицы взаимного перехода (АГЭСК -> ГСК)
163. **def** get\_transition\_matrix(S):
164. matrix **=** np.array([
165. [cos(S), sin(S), 0],
166. [**-**sin(S), cos(S), 0],
167. [0, 0, 1],
168. ])
169. **return** matrix

172. **def** find\_H(Ra, theta, i, big\_omega, omega, p, e, tau):
173. # АГЭСК
174. # ===============================================================================================================
175. Xa, Ya, Za **=** get\_agesk(theta**=**theta, ra**=**Ra, i**=**i, big\_omega**=**big\_omega, omega**=**omega)
176. AGESK\_COORD **=** np.array([[Xa], [Ya], [Za]])
177. # ===============================================================================================================
179. rate **=** 7.292115855306578e**-**05
180. t **=** get\_t\_from\_tau(tau**=**tau, theta**=**theta, p**=**p, e**=**e)
181. # ГСК
182. # =================================================
183. coord **=** get\_transition\_matrix(rate **\*** t).dot(AGESK\_COORD)
184. # =================================================
186. a **=** 6378.136
187. ez\_sqr **=** 0.0067385254
188. # Поиск B, L, H
189. # ==========================================================
190. x, y, z **=** coord[0][0], coord[1][0], coord[2][0]
191. D **=** math.hypot(x, y)
192. B, L, H **=** 0, 0, 0
193. **if** D **==** 0:
194. B **=** pi **/** 2 **\*** z **/** fabs(z)
195. L **=** 0
196. H **=** z **\*** sin(B) **-** a **\*** sqrt(1 **-** ez\_sqr **\*** sin(B) **\*\*** 2)
197. **else**:
198. La **=** math.asin(y **/** D)
199. **if** y < 0 **and** x > 0: L **=** 2 **\*** pi **-** La
200. **if** y < 0 **and** x < 0: L **=** pi **+** La
201. **if** y > 0 **and** x < 0: L **=** pi **-** La
202. **if** y > 0 **and** x > 0: L **=** La
203. **if** z **==** 0:
204. B **=** 0
205. H **=** D **-** a
206. **else**:
207. r **=** sqrt(x **\*\*** 2 **+** y **\*\*** 2 **+** z **\*\*** 2)
208. c **=** math.asin(z **/** r)
209. p **=** (ez\_sqr **\*** a) **/** (2 **\*** r)
210. s1 **=** 0
211. b **=** c **+** s1
212. s2 **=** math.asin((p **\*** sin(2 **\*** b)) **/** sqrt(1 **-** ez\_sqr **\*** sin(b) **\*\*** 2))
213. **while** **not** fabs(s2 **-** s1) < 1e**-**3:
214. s1 **=** s2
215. b **=** c **+** s1
216. s2 **=** math.asin((p **\*** sin(2 **\*** b)) **/** sqrt(1 **-** ez\_sqr **\*** sin(b) **\*\*** 2))
217. B **=** b
218. H **=** D **\*** cos(B) **+** z **\*** sin(B) **-** a **\*** sqrt(1 **-** ez\_sqr **\*** sin(B) **\*\*** 2)
220. **return** H
221. # ==========================================================

224. **def** find\_density(H):
225. table **=** None
227. **if** H <**=** 500:
228. table **=** table1
230. **if** 500 < H:
231. **if** H > 1500:
232. **pass**
233. # print(f'WARNING H={H}')
234. table **=** table2
236. index **=** table['F0'].index(Fa)
238. a0 **=** table['a0'][index]
239. a1 **=** table['a1'][index]
240. a2 **=** table['a2'][index]
241. a3 **=** table['a3'][index]
242. a4 **=** table['a4'][index]
243. a5 **=** table['a5'][index]
244. a6 **=** table['a6'][index]
246. density\_H **=** get\_density\_H(H**=**H, a0**=**a0, a1**=**a1, a2**=**a2, a3**=**a3, a4**=**a4, a5**=**a5, a6**=**a6)
247. **return** get\_density(density\_H**=**density\_H)

250. # Функция для вычисления плотности атмосферы
251. **def** get\_density(density\_H, K0**=**1, K1**=**0, K2**=**0, K3**=**0, K4**=**0):
252. **return** density\_H **\*** K0 **\*** (1 **+** K1 **+** K2 **+** K3 **+** K4)

255. # Функция для вычисления плотности ночной атмосферы
256. **def** get\_density\_H(H, a0, a1, a2, a3, a4, a5, a6):
257. density\_0 **=** 1.58868 **\*** 1e**-**8
258. power\_value **=** a0 **+** a1 **\*** H **+** a2 **\*** pow(H, 2) **+** a3 **\*** pow(H, 3) **+** a4 **\*** pow(H, 4) **+** a5 **\*** pow(H, 5) **+** a6 **\*** pow(H, 6)
259. **return** density\_0 **\*** math.exp(power\_value)
260. **import** json
261. **from** flask **import** Flask
262. **import** math
263. **from** functions **import** **\***
264. **from** time **import** sleep
265. **import** threading
266. **from** SqliteLogger **import** **\***
267. **import** os
268. os.chdir(os.path.dirname(os.path.realpath(\_\_file\_\_)))
270. **if** **not** os.path.exists(os.getcwd()**+**'\logs'):
271. os.mkdir('./logs')
272. print(os.getcwd())
273. d\_theta **=** 1e**-**2
275. DELAY\_IN\_SECS **=** .05
277. # h\_a = 1740
278. # h\_p = 350

281. h\_a **=** 2500
282. h\_p **=** 700
284. Earth\_radius **=** 6371
286. a **=** get\_a(ra**=**Earth\_radius **+** h\_a, rp**=**Earth\_radius **+** h\_p)
287. e **=** get\_e(ra**=**Earth\_radius **+** h\_a, rp**=**Earth\_radius **+** h\_p, a**=**a)
288. p **=** get\_p(a**=**a, e**=**e)
289. r **=** get\_r(p**=**p, e**=**e, theta**=**0)
291. flying\_objects **=** {
292. 'object1':
293. {
294. 'x': 0,
295. 'y': 0,
296. 'z': 0,
297. 'theta': 0,
298. 'h\_a': 1740,
299. 'h\_p': 350,
300. 'r': 6721.000000000001,
301. 'p': 7350.8671790722765,
302. 'OMEGA': math.radians(10),
303. 'omega': 0,
304. 'e': 0.09371628910463863,
305. 'U': math.radians(7),
306. 'tau': 0,
307. 'm': 1700,
308. 'i': math.radians(20.8)
309. },
311. 'object2':
312. {
313. 'x': 0,
314. 'y': 0,
315. 'z': 0,
316. 'theta': 0,
317. 'h\_a': 2500,
318. 'h\_p': 700,
319. 'r': 7071.0,
320. 'p': 7869.3816334211515,
321. 'OMEGA': math.radians(10),
322. 'omega': 0,
323. 'e': 0.11290929619872037,
324. 'U': math.radians(7),
325. 'tau': 0,
326. 'm': 1700,
327. 'i': math.radians(47)
328. }
330. }

333. **def** loop():
334. database\_name **=** get\_unique\_name()
335. conn **=** sqlite3.connect('./logs/'**+**database\_name)
336. cursor **=** conn.cursor()
337. **while** True:
338. **for** key **in** flying\_objects.keys():
339. # Создание таблицы в Sqlite3
340. # ========================================
341. create\_query **=** get\_create\_table\_query(key)
342. cursor.execute(create\_query)
343. conn.commit()
344. # ========================================
346. r1 **=** p1 **=** OMEGA1 **=** omega1 **=** i1 **=** e1 **=** tau1 **=** 0
347. r **=** flying\_objects[key]['r']
348. p **=** flying\_objects[key]['p']
349. OMEGA **=** flying\_objects[key]['OMEGA']
350. omega **=** flying\_objects[key]['omega']
351. i **=** flying\_objects[key]['i']
352. e **=** flying\_objects[key]['e']
353. tau **=** flying\_objects[key]['tau']
354. theta **=** flying\_objects[key]['theta']
356. S **=** T **=** W **=** 0
357. F **=** geT\_F(r**=**r, S**=**S, T**=**T, theta**=**theta, e**=**e, p**=**p)
359. p1 **=** p **+** R\_p(r**=**r, T**=**T, F**=**F) **\*** d\_theta
360. OMEGA1 **=** OMEGA **+** R\_OMEGA(W**=**W, F**=**F, r**=**r, p**=**p, u**=**theta **+** omega, i**=**i) **\*** d\_theta
361. i1 **=** i **+** R\_i(W**=**W, F**=**F, r**=**r, p**=**p, u**=**theta **+** omega) **\*** d\_theta
362. omega1 **=** omega **+** R\_omega(F**=**F, S**=**S, theta**=**theta, e**=**e, T**=**T, r**=**r, p**=**p, W**=**W, i**=**i, u**=**theta **+** omega) **\*** d\_theta
363. e1 **=** e **+** R\_e(F**=**F, S**=**S, theta**=**theta, T**=**T, r**=**r, p**=**p, e**=**e) **\*** d\_theta
364. tau1 **=** tau **+** R\_tau(F**=**F, p**=**p) **\*** d\_theta
366. theta **+=** d\_theta
367. r1 **=** get\_r(p**=**p1, e**=**e1, theta**=**theta)
369. x, y, z **=** get\_agesk(theta**=**theta, ra**=**r1, i**=**i1, big\_omega**=**OMEGA1, omega**=**omega1)
370. flying\_objects[key]['r'] **=** r1
371. flying\_objects[key]['p'] **=** p1
372. flying\_objects[key]['OMEGA'] **=** OMEGA1
373. flying\_objects[key]['omega'] **=** omega1
374. flying\_objects[key]['i'] **=** i1
375. flying\_objects[key]['e'] **=** e1
376. flying\_objects[key]['tau'] **=** tau1
377. flying\_objects[key]['theta'] **=** theta
378. flying\_objects[key]['x'] **=** x
379. flying\_objects[key]['y'] **=** y
380. flying\_objects[key]['z'] **=** z
381. m **=** flying\_objects[key]['m']
383. # Фиксация итерации в таблице Sqlite3
384. # ==============================================================================
385. insert\_query **=** get\_insert\_query(key, x, y, z, theta, r1, p1, OMEGA1, omega1, e1, tau1, m, i1)
386. # insert\_query.format(key, x, y, z, theta, r1, p1, OMEGA1, omega1, e1, tau1, m, i1)
387. cursor.execute(insert\_query)
388. conn.commit()
389. # ==============================================================================
391. sleep(DELAY\_IN\_SECS)

394. loop\_thread **=** threading.Thread(target**=**loop)
396. app **=** Flask(\_\_name\_\_)

399. @app.route('/')
400. **def** index():
401. **return** json.dumps(flying\_objects)

404. **if** \_\_name\_\_ **==** "\_\_main\_\_":
405. loop\_thread.start()
406. app.run(host**=**'0.0.0.0')
407. **import** sqlite3
408. **import** datetime

411. **def** get\_create\_table\_query(table\_name):
412. query **=** "create table if not exists {}" \
413. "(x REAL, y REAL, z REAL, theta REAL, " \
414. " r REAL, p REAL, big\_omega REAL, " \
415. "omega REAL, e REAL, tau REAL, m REAL, i REAL );"
416. **return** query.format(table\_name)

419. **def** get\_insert\_query(key, x, y, z, theta, r1, p1, OMEGA1, omega1, e1, tau1, m, i1):
420. query **=** f"""INSERT INTO {key} VALUES ({x},{y},{z},{theta},{r1},{p1},{OMEGA1},{omega1},{e1},{tau1},{m},{i1})"""
421. **return** query

424. **def** get\_unique\_name():
425. now **=** datetime.datetime.now()
426. day **=** str(now.day).rjust(2, '0')
427. month **=** str(now.month).rjust(2, '0')
428. year **=** str(now.year)
429. hour **=** str(now.hour).rjust(2, '0')
430. minute **=** str(now.minute).rjust(2, '0')
431. second **=** str(now.second).rjust(2, '0')
432. result **=** f'{hour}{minute}{second}{day}{month}{year}.db'
433. **return** result

436. **def** make\_new\_connection():
437. database\_name **=** get\_unique\_name()
438. connection **=** sqlite3.connect()

441. print(get\_create\_table\_query('test'))