Федеральное государственное автономное образовательное учреждение высшего образования
«Российский университет дружбы народов имени Патриса Лумумбы»
Инженерная академия
Департамент механики и процессов управления

ОТЧЕТ

По Лабораторной работе №1 по Механике Космического Полета.
Вариант 4.

Направление: 01.03.02 Прикладная математика и информатика
(код направления / название направления)

Профиль: Математические методы механики полета раке-носителей и космических аппаратов
(название профиля)

Тема: Уравнение Кеплера для эллиптической орбиты
(название лабораторной / курсовой)

Выполнено студентом: Критским Матвеем Димитриевичем
(ФИО)

Группа: ИПМбд-02-22
№ студенческого: 1132226149

Москва, 2023

Благодаря миссии "Луна-10" ученые установили несферичность гравитационного потенциала Луны и впервые получили его точную модель. Также было уточнено значение массы Луны.

Данные миссии "Луна-10":

μ(Гравитационная постоянная Луны) = 4902.8000 Н
R(Радиус Луны) = 1737.1 км;
R_a(радиус-вектор апоцентра) = R + 1017 = 2754.1 км
R_p(радиус-вектор перицентра) = R + 350 = 2087.1 км
a(большая полуось) = (R_a + R_p) / 2 = 2420.6 км
e(эксцентриситет) = (R_a - R_p) / (R_a + R_p) = 0.137776
T(период обращения) = 2 часа 58 минут 15 секунд
PI(число Пи) = 3.1415926

ИЛИ

10695 секунд

Формулы для подсчета аномалий:

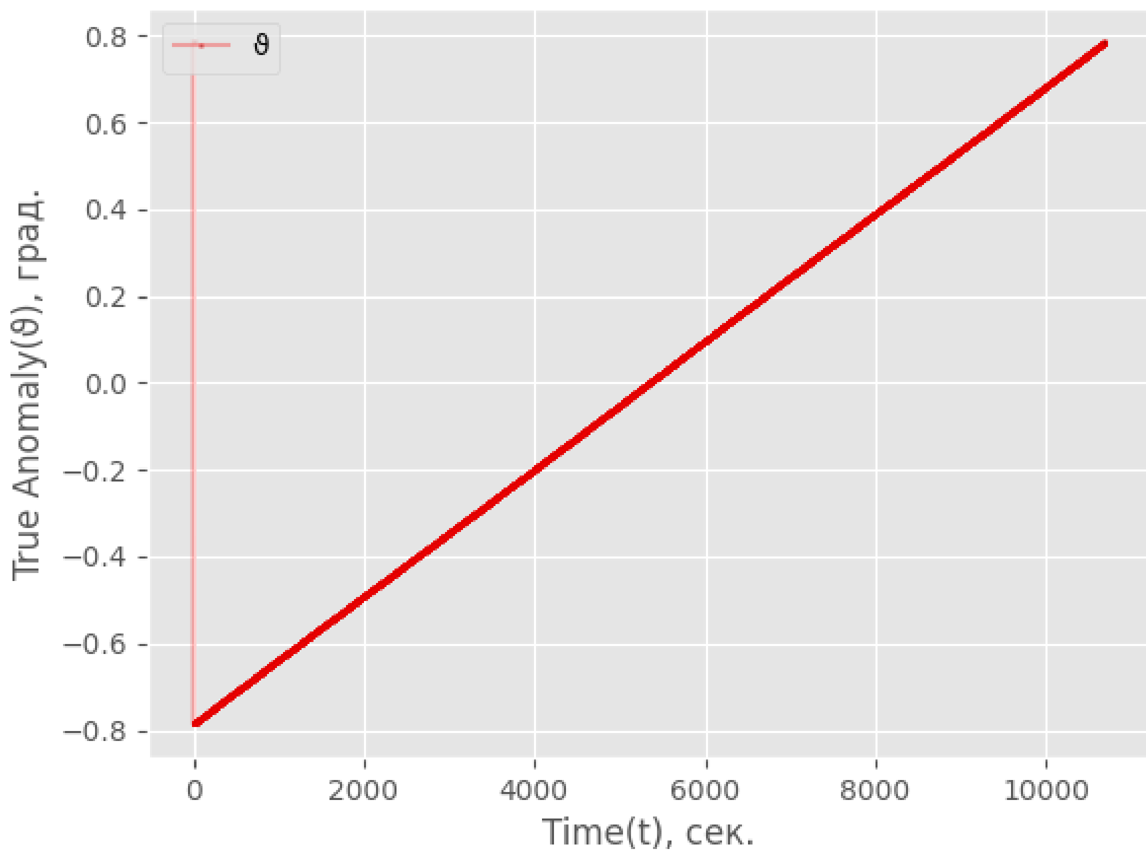
M(t) = (2 * PI / T) * t

E(t) = M + e * sin(past_E) - метод последовательных итераций, где past_E - предыдущая итерация, а E - текущая. Начальная итерация равна нулю.

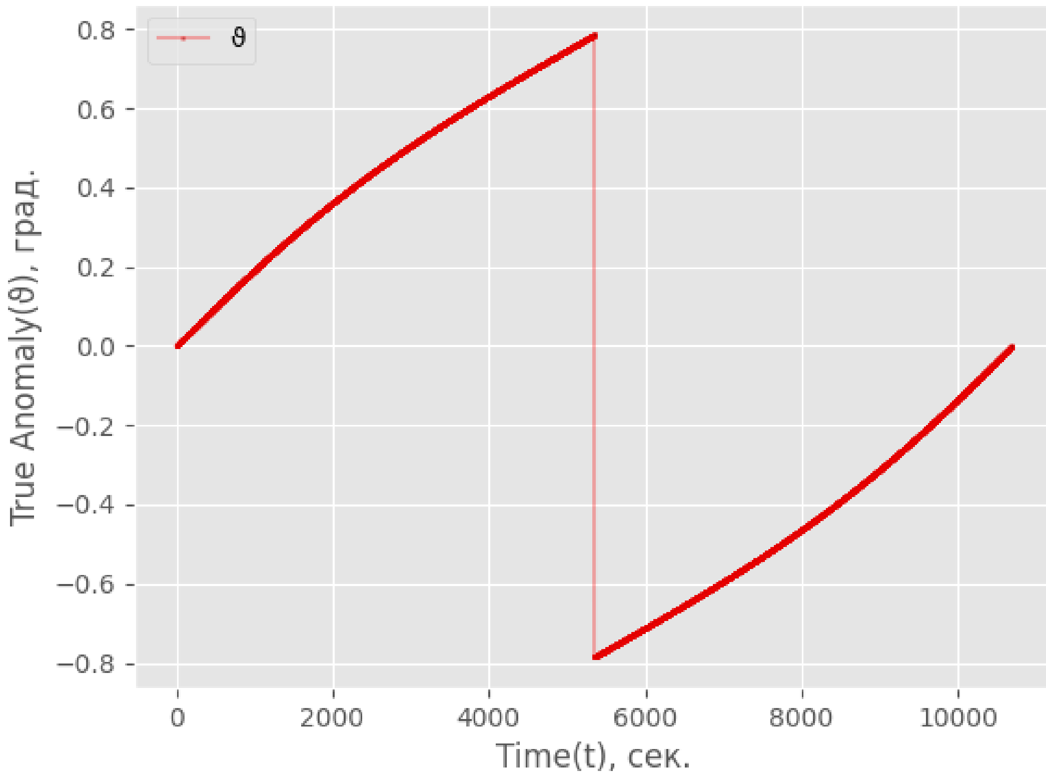
ϑ(t) = atan(tan((E + PI) / 2) * sqrt((1 + e) / (1 - e))) / 2

Графикb зависимостей аномалий от времени t:

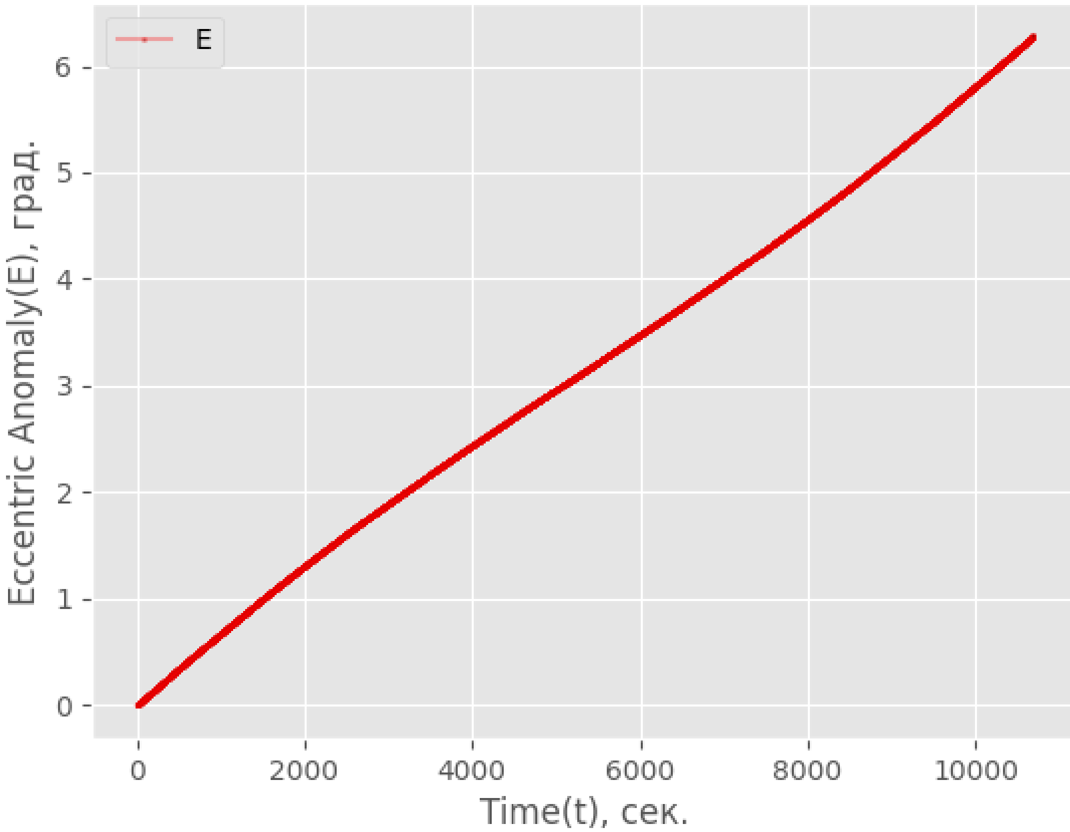
Истинной (ϑ(t)): (график тангенса со смещением на PI/2) (из-за огромного масштаба похож на прямую)



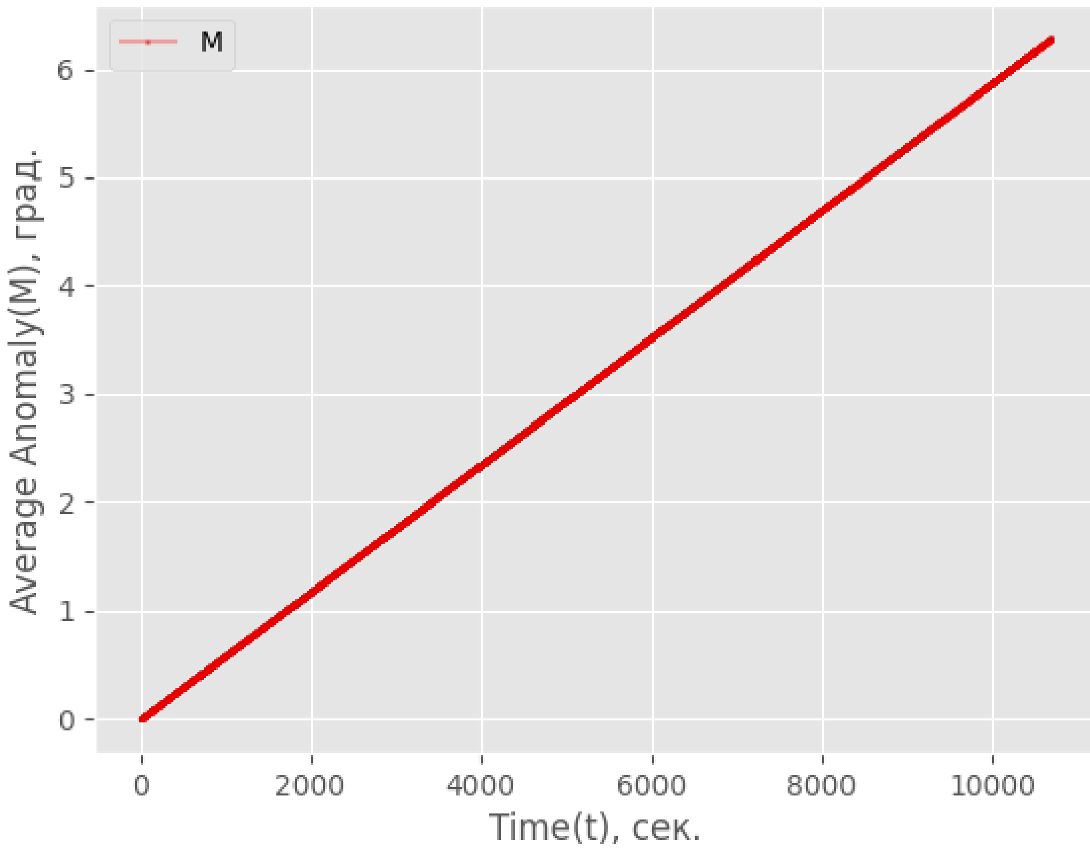
Этот же график, но без смещения:



Эксцентрической E(t): (график тангенса)



Средней M(t): (график прямой)



Различные методы нахождения эксцентрической аномалии расписаны в программе METHODS.c, тогда как код для подсчета предыдущих значений и последующих находится в файле task_1.c (см. приложение).

Зная истинную аномалию ϑ и факальный параметр p, можно подсчитать радиус-вектор r, нормальную скорость V_n, тангенциальную скорость V_r и скорость тела V.

Используемые формулы:

$p = (R_a - R_p) / (2 * a) = 0.4879$ км

$r = p / (1 + e * \cos(\vartheta))$ км

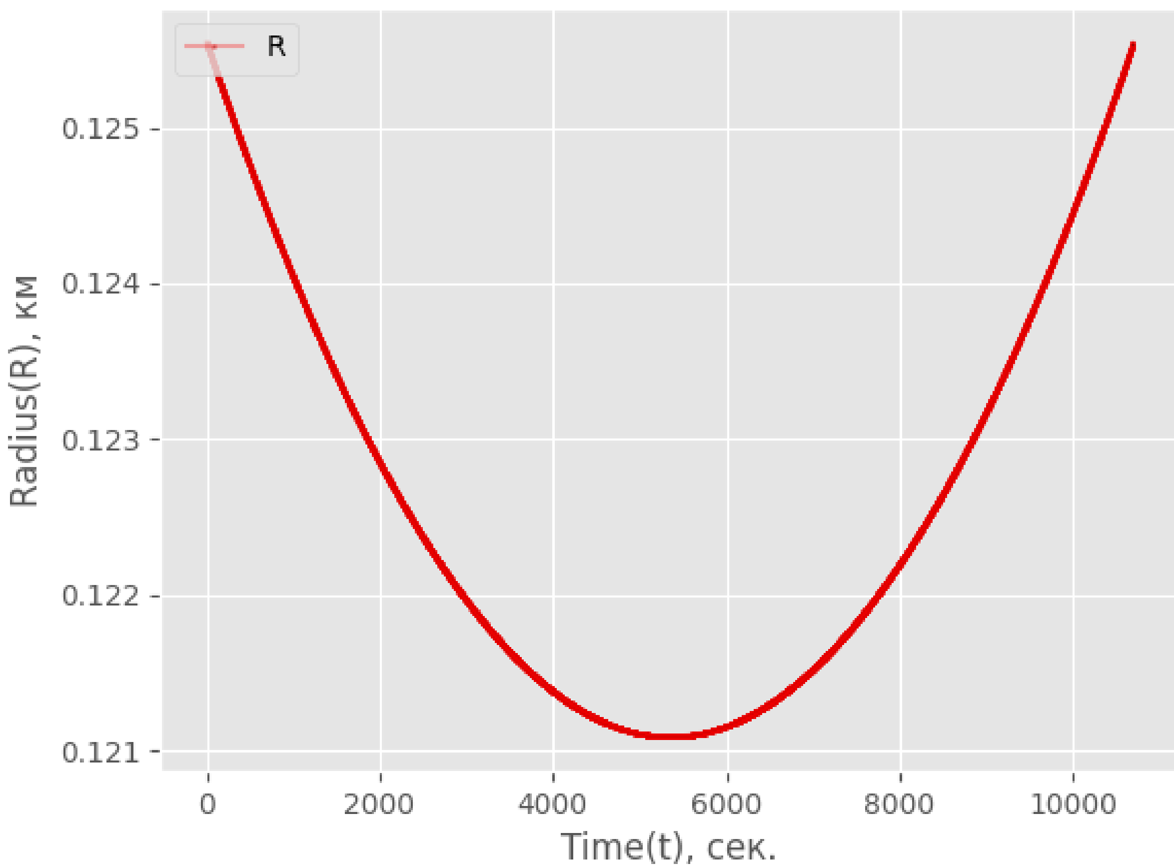
$V_n = \mathrm{sqrt}(\mu / p) * (1 + e * \cos(\vartheta))$ км/с

$V_r = \mathrm{sqrt}(\mu / p) * e * \sin(\vartheta)$ км/с

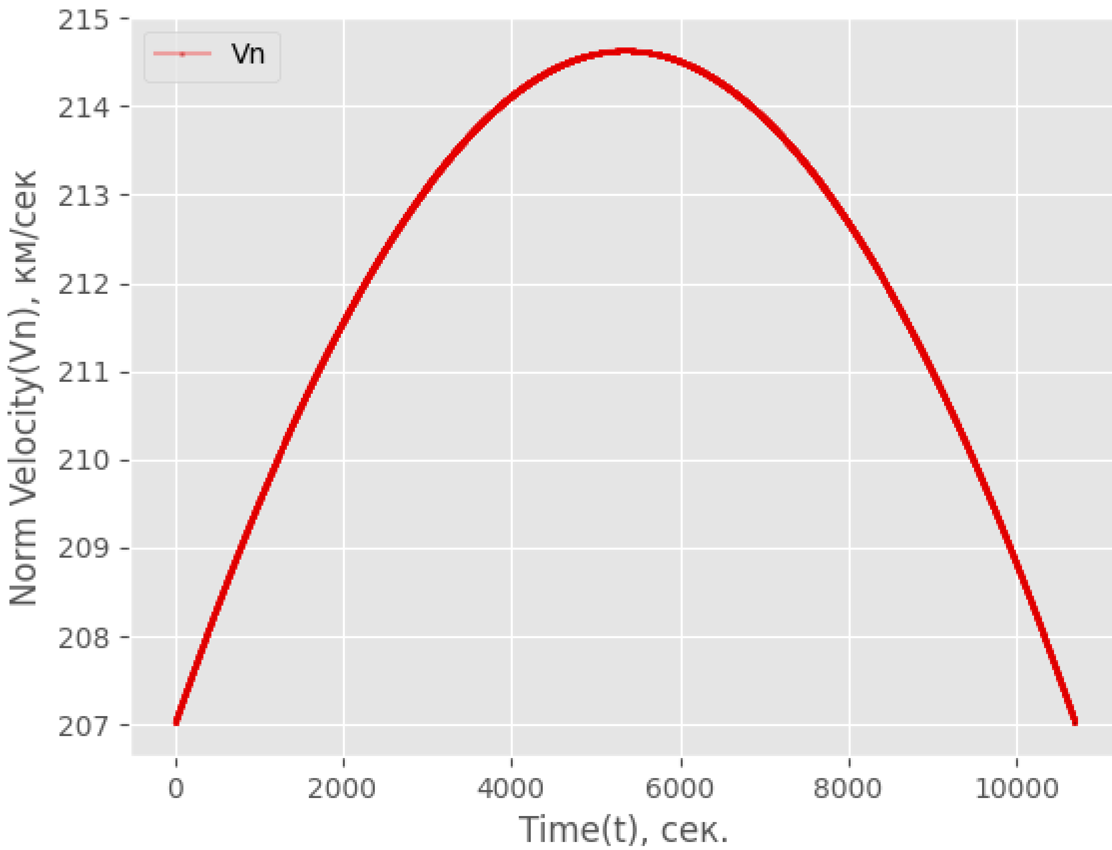
$V = \mathrm{sqrt}(V_r^2 + V_n^2)$ км/с

Графики зависимостей:

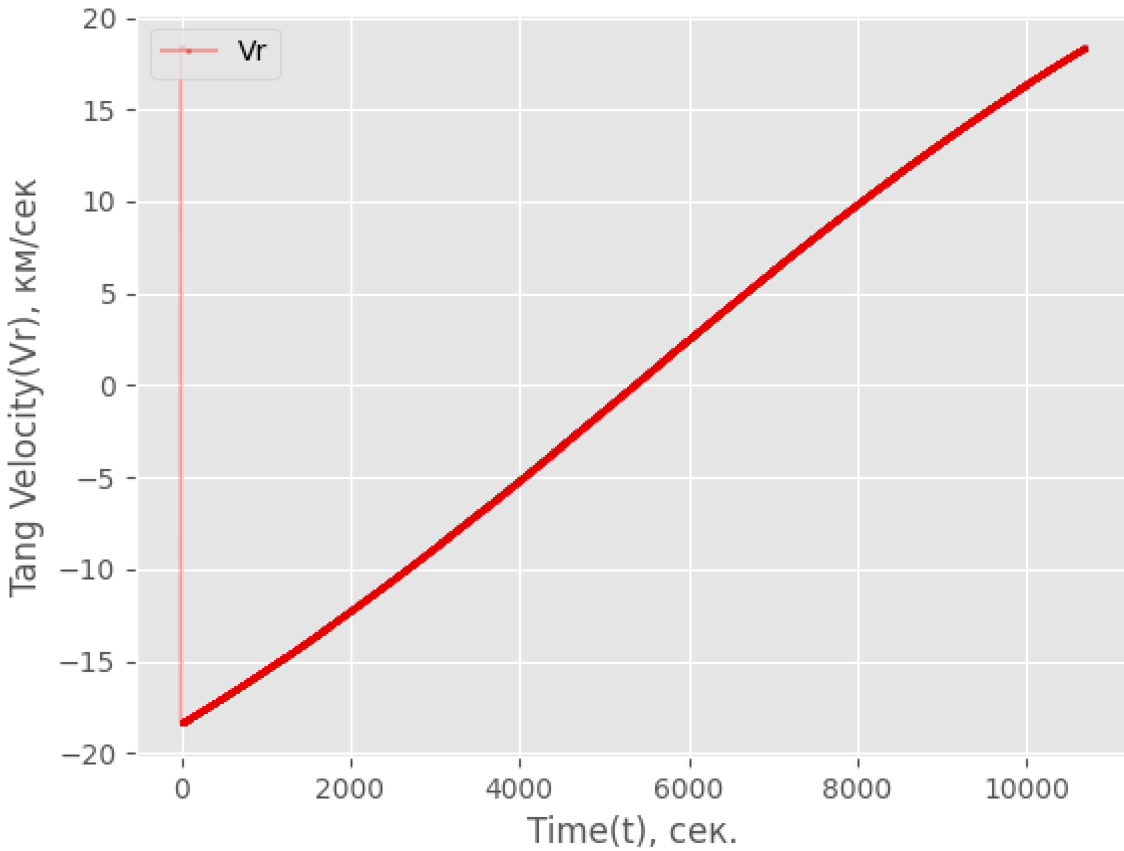
Радиус-вектора от t r(t): (парабола)



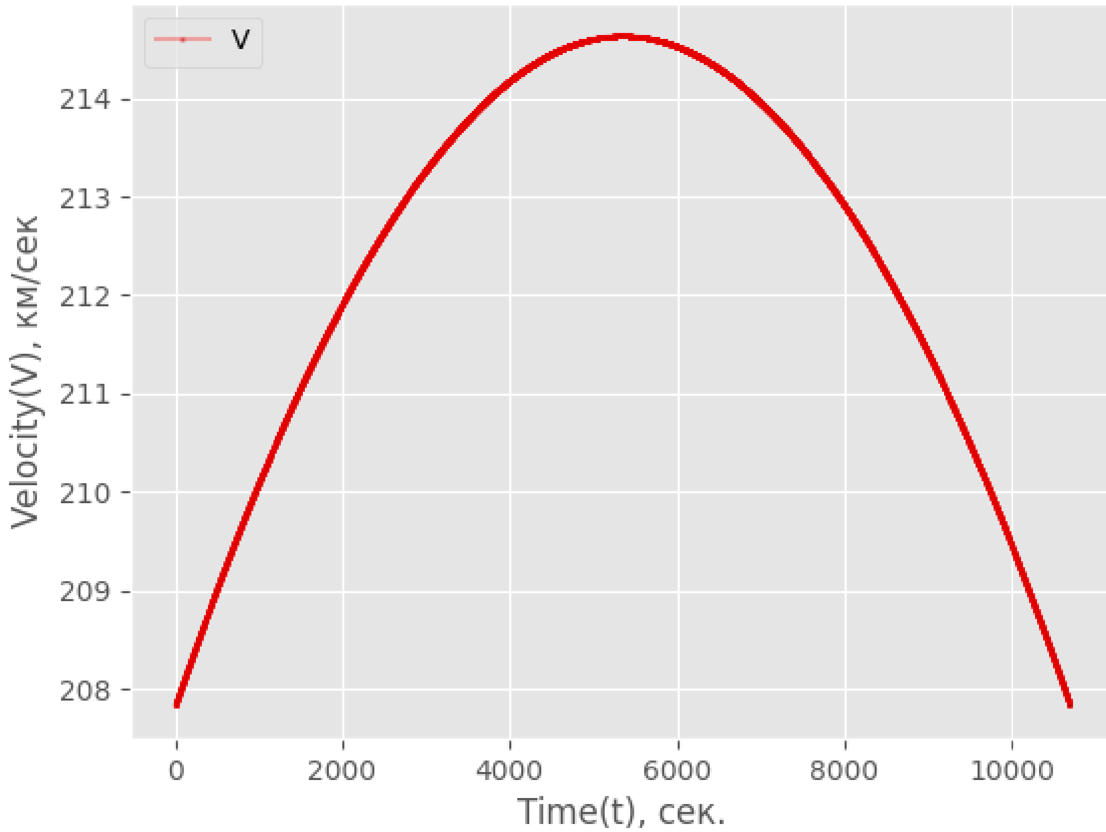
Нормальной скорости от t $V_n(t)$: (парабола)



Тангенциальной скорости от t $V_r(t)$:



Общей скорости от t V(t): (парабола)



Приложение:
Ссылка на код в GitHub: https://github.com/rudnmk/MSF/tree/main/1

Код программы:
METHODS.c:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define SIGMA 0.001

float* IterationMethod(float, int);
float* HalfDivMethod(float, float, float, int);
float* GoldenRatioMethod(float, float, float, int);
float* NewtonMethod(float, int);
```

```
float* LaguerreMethod(float, int);


int main() {
    float Ra = 1017.0;
    float Rp = 350.0;
    float a = (Ra + Rp) / 2;
    float e = (Ra - Rp) / (Ra + Rp);
    float p = (Ra - Rp) / 2 * a;
    float T = 10695.0;
}


float* IterationMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    E_ARR[0] = 0;
    for(int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        E_ARR[i] = e * sin(E_ARR[i - 1]) + M;
    }

    return E_ARR;
}

float* HalfDivMethod(float e, float A, float B, int T) {
    int i = 0;
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int flag = 0;
    float C;

    while ((i < T) && ((B - A) > SIGMA)) {
        float M = (2 * 3.14 / T) * i;
        if (flag == 0) {
            C = (B + A) / 2;
        }

        else {
            C = (B - A) / 2;
        }
```

```
        float Ea = A - e * sin(A) - M;
        float Eb = B - e * sin(B) - M;
        float Ec = C - e * sin(C) - M;

        if (Ea * Ec <= 0.0) {
            B = C;
        }
        else {
            A = C;
        }

        if ((A < 0.0 && B < 0.0) || (A > 0.0 && B > 0.0)) {
            flag = 1;
        }

        E_ARR[i] = Ec;
        i++;

        if (E_ARR[i] == E_ARR[i - 1]) {
            break;
        }
    }

    return E_ARR;
}

float* GoldenRatioMethod(float e, float A, float B, int T) {
    int i = 0;
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int flag = 0;
    float C;

    while ((i < T) && ((B - A) > SIGMA)) {
        int M = (2 * 3.14 / T) * i;
        if (flag == 0) {
            C = (B + A) / 1.618;
        }

        else {
            C = (B - A) / 1.618;
```

```
        }

        float Ea = A - e * sin(A) - M;
        float Eb = B - e * sin(B) - M;
        float Ec = C - e * sin(C) - M;

        if (Ea * Ec <= 0) {
            B = C;
        }
        else {
            A = C;
        }

        if ((A < 0 && B < 0) || (A > 0 && B > 0)) {
            flag = 1;
        }

        E_ARR[i] = Ec;
        i++;
    }

    return E_ARR;
}

float* NewtonMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    E_ARR[0] = 0;
    for (int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        float fE = E_ARR[i - 1] - e * sin(E_ARR[i - 1]) - M;
        float DfE = 1 - e * cos(E_ARR[i - 1]);
        E_ARR[i] = E_ARR[i - 1] - (fE / DfE);
    }
    return E_ARR;
}

float* LaguerreMethod(float e, int T) {
    float* E_ARR = (float*)malloc(T * sizeof(float));
    int n = 3;
    E_ARR[0] = 0;
```

```
    for (int i = 1; i < T; i++) {
        float M = (2 * 3.14 / T) * i;
        float fE = E_ARR[i - 1] - e * sin(E_ARR[i - 1]) - M;
        float DfE = 1 - e * cos(E_ARR[i - 1]);
        float DDfE = e * sin(E_ARR[i - 1]);
        float hE = fabs((n - 1) * ((n - 1) * pow(DfE, 2) - n * fE * DDfE));
        E_ARR[i] = E_ARR[i - 1] - ((fE * n) / (DfE + sqrt(hE)));
    }
    return E_ARR;
}
```

----

task_1.c:

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define PI 3.1415926
#define μ 4902.8000  //Гравитационный параметр Луны

int DataCollection(float, float, float);


int main() {
    float Ra = 2754.1;
    float Rp = 2087.1;
    float a = (Ra + Rp) / 2;
    float e = (Ra - Rp) / (Ra + Rp);
    float p = (Ra - Rp) / (2 * a);
    DataCollection(a, e, p);
    return 0;
}

int DataCollection(float a, float e, float p) {
    FILE* T_ = fopen("T.txt", "w");
    FILE* M_dat = fopen("M_Data.txt", "w");
    FILE* E_dat = fopen("E_Data.txt", "w");
    FILE* THETA_dat = fopen("THETA_Data.txt", "w");
    FILE* R_dat = fopen("Radius_Data.txt", "w");
```

```
    FILE* Vn_dat = fopen("VelN_Data.txt", "w");
    FILE* Vr_dat = fopen("VelR_Data.txt", "w");
    FILE* V_dat = fopen("Velocity_Data.txt", "w");

    float past_E = 0.0;
    //float T = 2 * PI * sqrt(pow(a, 3) / μ);
    float T = 10695.0;
    fprintf(T_, "%f", T);
    fclose(T_);
    for (float i = 0.0; i < T; i++) {
        float M = (2 * PI / T) * i;
        float E = M + e * sin(past_E);
        float THETA = atan(tan((E + PI) / 2) * sqrt((1 + e) / (1 - e))) / 2;
        float radius = p / (1 + e * cos(THETA));
        float Vn = sqrt(μ / p) * (1 + e * cos(THETA));
        float Vr = sqrt(μ / p) * e * sin(THETA);
        float V = sqrt(pow(Vn, 2.0) + pow(Vr, 2.0));

        fprintf(M_dat, "%f \n", M);
        fprintf(E_dat, "%f \n", E);
        fprintf(THETA_dat, "%f \n", THETA);
        fprintf(R_dat, "%f \n", radius);
        fprintf(Vn_dat, "%f \n", Vn);
        fprintf(Vr_dat, "%f \n", Vr);
        fprintf(V_dat, "%f \n", V);

        past_E = E;
    }


    fclose(M_dat);
    fclose(E_dat);
    fclose(THETA_dat);
    fclose(R_dat);
    fclose(Vn_dat);
    fclose(Vr_dat);
    fclose(V_dat);
}
```