Домашнее задание
по предмету «Архитектуре ЭВМ»
Федяков Филипп,
студент группы ИУ9-11

20.10.2014 — 14.01.15

Постановка задачи:

Задача: Разработать подпрограмму численного моделирования движения тел (материальных точек) в трёхмерном пространстве в гравитационном поле, созданном очень массивным телом, по сравнению с которым можно пренебречь массой других тел и их взаимодействием между собой. Действующая на тело сила Fi = -ri·K·mi/|ri|3, где ri - радиус-вектор, определяющий положение тела i в пространстве.
Ускорение тела i: ai = Fi/mi = -ri·K/|ri|3.
Для интегрирования дифференциальных уравнений применять:
метод Эйлера (Euler Scheme):
В методе Эйлера предполагается, что ускорение тела at в момент времени t зависит только от положения тела в пространстве: at=f(rt), тогда положение и скорость тела в следующий момент времени t+Δt может быть определено как
vt+Δt = vt + at·Δt,
rt+Δt = rt + vt·Δt + ½·at·Δt2,
где r, v и a - радиус-вектор, скорость и ускорение в моменты времени t и t+Δt, взятые с шагом Δt
Дополнительные указания-использовать тип данных double;

Первая версия программы (наивная реализация):

```
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
double K = 0; //39.856e+13;
double R = 6.371e+6;//1.3e+17;

struct vector {
    double x;
    double y;
    double z;
};

void Assign (struct vector *a, struct vector b) {
    *a = b;
}

void Accel_for_structs (struct vector *r, struct vector *a) {
    double r3;
    r3 = sqrt(r->x * r->x + r->y * r->y + r->z * r->z);
    if (r3 == 0) {
    a->x = 0;
    a->y = 0;
    a->z = 0;
    }
    else
    r3 = - K / (r3 * r3 * r3);
    a->x = r->x * r3;
    a->y = r->y * r3;
    a->z = r->z * r3;
}
```

```
void Euler_for_structs (struct vector *r, struct vector *v, double dt, int m) {
    int i, j;
    double dt2 = dt*dt/2;
    struct vector a;
    for (i = 1; i <= m; i++) {
    Accel_for_structs(r, &a);
    r->x = r->x + v->x * dt + a.x * dt2;
    r->y = r->y + v->y * dt + a.y * dt2;
    r->z = r->z + v->z * dt + a.z * dt2;
    v->x = v->x  + a.x * dt;
    v->y = v->y  + a.y * dt;
    v->z = v->z  + a.z * dt;
    }
}

double sqr(double x) { return x*x; }

int main () {
    int i, j, n;
    double t = 10;
    struct vector r1,r0, v1, v0, a;
    r0.x = 0;
    r0.y = 0;
    r0.z = 0;
    v0.x = 0;
    v0.y = 0;
    v0.z = 0;
    double real_x = 500;
    double dt;
    for (n = 1; n <= 1000; n += 1 + (n/50) + 2*(n/200) ) {
    dt = t / n;
    Assign(&r1, r0);
    Assign(&v1, v0);
    Euler_for_structs(&r1, &v1, dt, n);
    printf("%d;%.14G\n", n, (r1.x-real_x) / real_x);
    }
    return 0;
}
```

Общие сведения о решении и поставленной задаче:

K = G*M — константа, где G — гравитационная постоянная, M — масса планеты.

t — время интегрирования.

N — количество шагов.

$$dt = \frac{t}{n}$$ - шаг по времени

$r_0 (x_0 ; y_0 ; z_0)$ - начальный радиус вектор положения от начала координат, связанное с центром планеты

$v_0 (v_{0X} ; v_{0Y} ; v_{0Z})$ - начальный радиус вектор скорости.

$r_{real} (x_{real} ; y_{real} ; z_{real})$ - конечное положение тела (вычисленное аналитически).

$r_n (x_n ; y_n ; z_n)$ - конечное положение тела (вычисленное с помощью программы).

E — относительная ошибка

$$E = \frac{r_{real} - r_n}{r_{real}}$$

Машинную точность будем обозначать буквой e.

В наших вычислениях мы часто будем пользоваться формулой:

$$a(1+e)^b = a(1+be), \text{где } e \ll 1$$

Тестирование первой версии:

Введем нашу систему тестов, к которой мы потом будем часто обращаться. Замечу, что если в условии теста ничего не сказано про некоторые начальные данные, значит они не используются в данном тесте

Тест 1

Описание: поворот вокруг земли на $1.2414*10^{-1}$ радиан в плоскости xOy

K = $39.856*10^{13}$, что соответствует Земной массе, умноженную на гравитационную постоянную.

R = $6.371* 10^6$ - радиус земли.

Начальное положение: r_0 = (R ; 0 ; 0)

Начальная скорость: v_0 = (0 ; sqrt (g*R); 0), где g посчитано через K в программе.

t = 100 секунд

Точные результаты :

x_{real} = $0.7874*10^6$

y_{real} = $6,322*10^6$

$$E = \frac{\sqrt{(x_n - x_{real})^2 + (y_n - y_{real})^2}}{R}$$

К сожалению, при первой неудачной обработке данных все значения были округлены в научном представлении до двух знаков после запятой, что немного задержало исследование: на графике появились горизонтальные линии. Много времени ушло на то, чтобы понять причину их появления.

Исправленные результаты представлены ниже (см. рисунок 1).

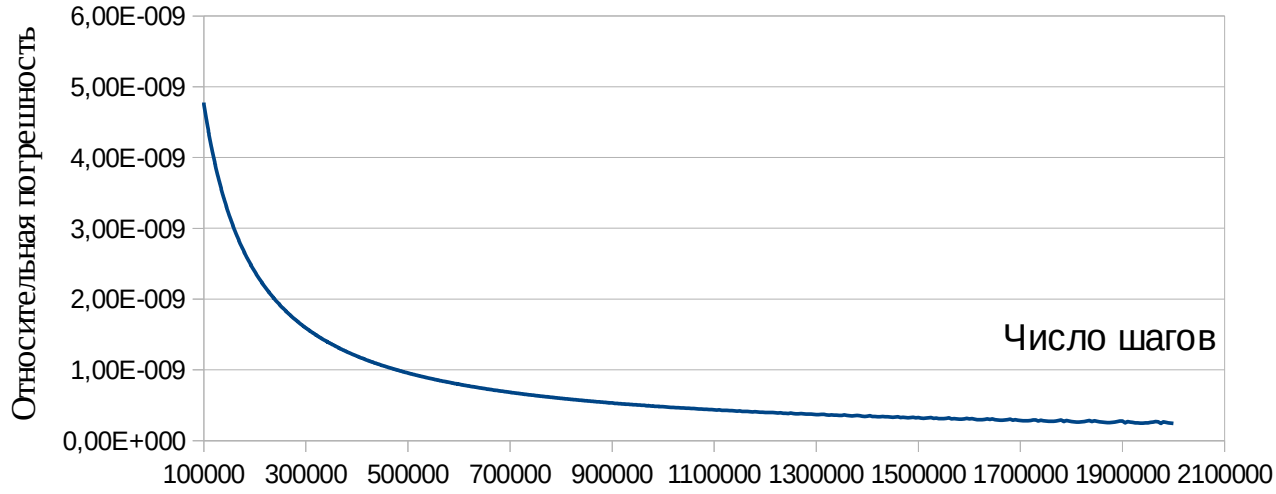

Рисунок 1: Тест 1 версии 1

Наблюдается стабильное поведение графика.
«Проблемы» появились только при числе шагов n > 900 000.
Масштабированная часть графика представлена ниже (см рисунок 2).

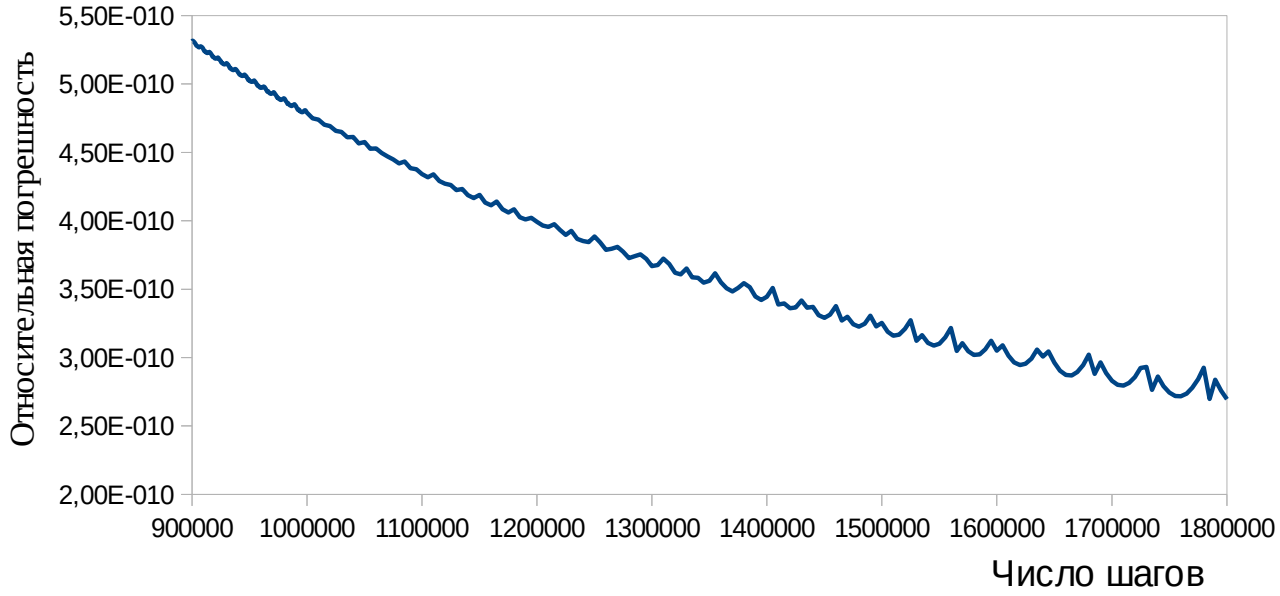

Рисунок 2: Тест 1 версии 1 (увеличенный масштаб)

Было принято решение провести тесты на линейное движение тела, без влияния других массивных тел.

Тест 2

Описание: линейное движение тела, без ускорения

K = 0;

r_0 = (0 ; 0 ; 0)

v_0 = (10 ; 0 ; 0)

t = 10

x_{real} = 100

Относительная погрешность:

$$E = \frac{x_{real} - x}{x}$$

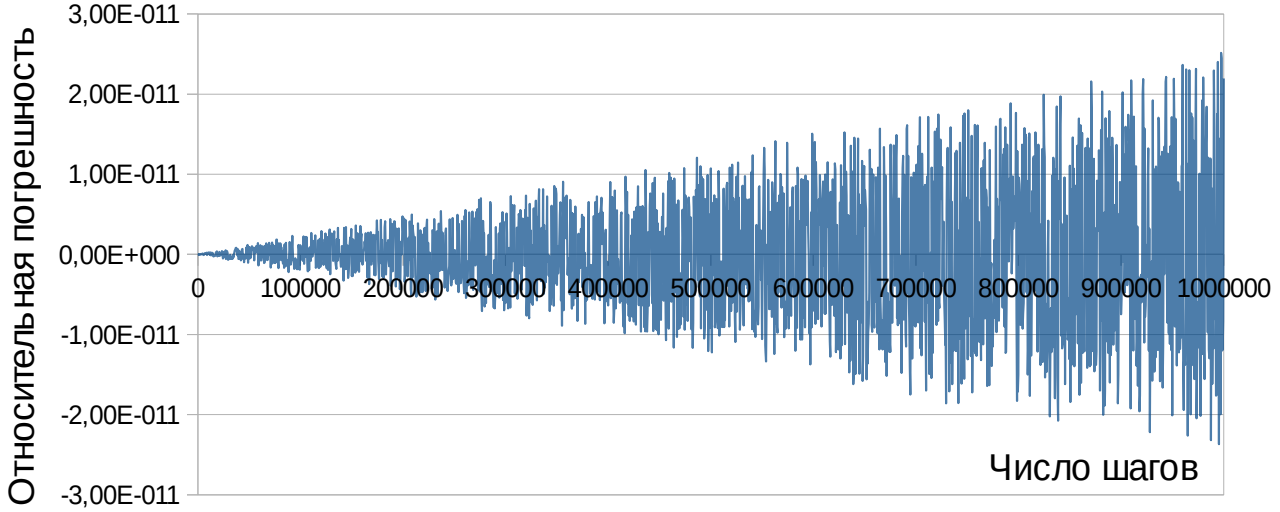

Рисунок 3: Тест 2 версии 1

По полученным результатам (см. рис. 3) были выдвинуты несколько задач будущего исследования:
- проверить на линейное накопление погрешности в результате суммировании
- исследовать зависимость погрешности от размера dt (dt = t / n)
- исследовать влияние начальных входных данных на погрешность

Во время всех следующих исследований примем a = const. Позже при необходимости мы вычислим и ее погрешность.

Гипотеза 1 - погрешность при вычислении dt

Вычислим погрешность при движении вдоль одной оси.

Погрешность при вычислении скорости:

$$dt=\frac{t}{n}(1+e), \text{где e - погрешность} (e=\pm\partial)$$

$$v_1=v_0+adt(1+e),$$

Тогда:

$$v_2=v_1+adt(1+e)=v_0+2\text{a}\,dt(1+e)$$

Получаем:

$$v_i=v_0+i\,a\cdot dt(1+e)$$

Погрешность при вычислении координаты:

$$x_1=x_0+v_{0\text{dt}}(1+e)+\frac{adt^2}{2}(1+e)^2$$

$$x_2=x_1+v_{1\text{dt}}(1+e)+\frac{adt^2}{2}(1+e)^2$$

$$x_2=x_0+v_0dt(1+e)+\frac{adt^2}{2}(1+e)^2+v_1dt(1+e)+\frac{adt^2}{2}(1+e)^2 x_2=x_0+v_0\,dt(1+e)+v_1\,dt(1+e)+2\frac{adt^2}{2}(1+2e)$$

Тогда получаем:

$$x_i=x_0+dt(1+e)\sum_{j=0}^{j=i-1}v_j+i\frac{adt^2}{2}(1+2e)$$

Подставляем найденные выше значения скоростей в сумму и получаем (используя сумму арифметической прогрессии):

$$dt(1+e)\sum_{j=0}^{j=i-1}v_i=dt(1+e)(v_0+v_0+adt(1+e)+v_0+2\text{adt}(1+e)+..+v_0+(i-1)adt(1+e))=$$

$$=iv_0dt(1+e)+i\frac{adt^2}{2}(i-1)(1+2e)$$

Тогда в итоге:

$$x_n=x_0+n\,v_0\,dt(1+e)+n\frac{a\,dt^2}{2}(n-1)(1+2e)+n\frac{a\,dt^2}{2}(1+2e)$$

$$x_n=x_0+nv_0\,dt(1+e)+n^2\frac{adt^2}{2}(1+2e)$$

Подставляем dt = t / n:

$$x_n=x_0+v_0t(1+e)+\frac{at^2}{2}(1+2e)$$

$$x_n-x_{real}=e\,v_0t+2e\frac{at^2}{2}$$

$$E=\frac{e\,v_0t+eat^2}{x_0+v_0t+at^2/2}$$

Проверка 1:
Для проверки гипотезы используем тест 2. Тогда очевидно, что погрешность от неточного вычисления dt будет константой.
Проверим нашу гипотезу, вручную округлив dt до 12 знака после запятой. Тогда по нашей формуле (см. выше в этом параграфе) относительная ошибка должна иметь порядок 10^{-12} при малом N — число шагов, когда влияние погрешности при суммировании будет крайне мала.
Функция округления:

```
double my_round(double x, int i) {
    x *= pow(10, i);
    x = round(x);
    x /= pow(10, i);
    return x;
}
```

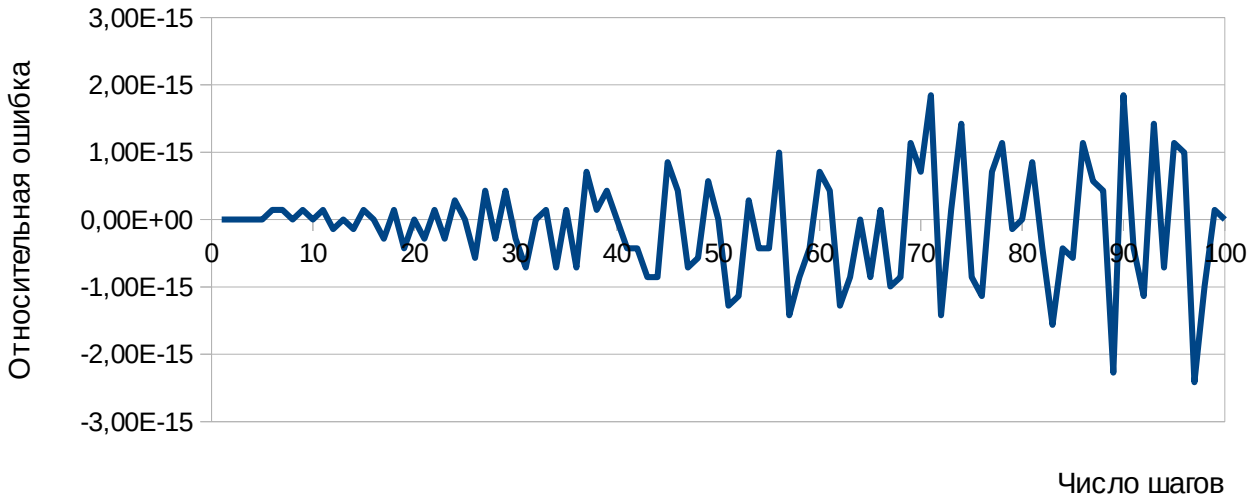

Рисунок 4: Тест 2 версии 1 без округления dt

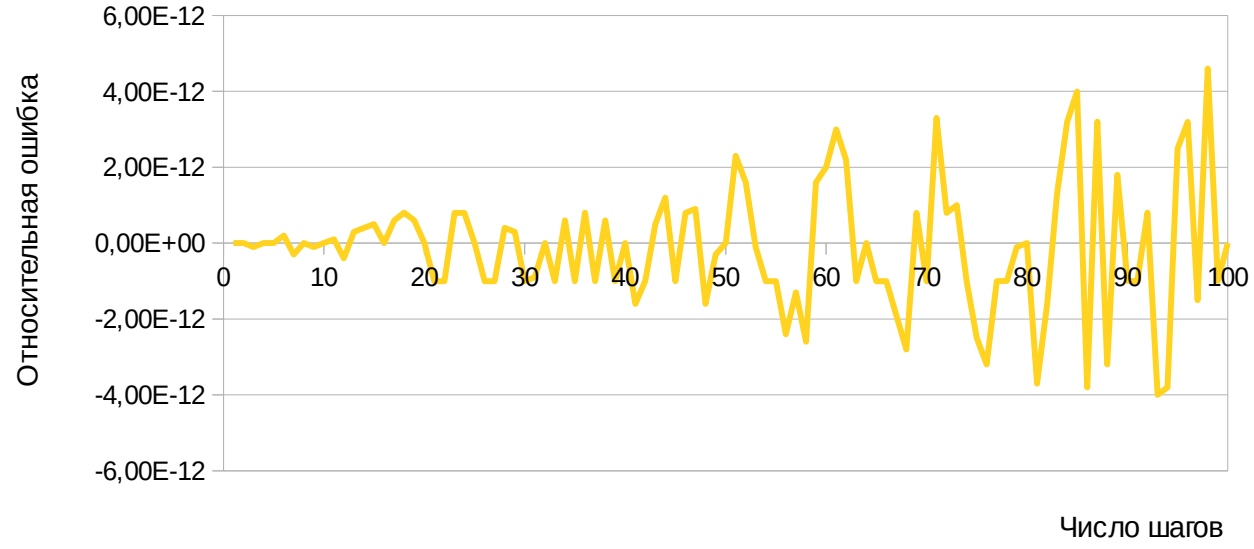

Рисунок 5: Тест 2 версии 1 с округлением dt

Полученные результаты отобразим на графиках (см. рис. 4 и рис. 5):
Как мы видим, результаты не оправдали наших ожиданий, так как график начинается вовсе не из нуля, более того, он растет линейно.
Скорее всего это ошибка работы функции my_round.
Поэтому мы проведем другой способ проверки — приведение к типу float.
Проверка 2:
Единственные изменения, которые мы внесем в программу — dt и t теперь объявлены как float.
Для проверки опять же используем тест 2. Мы ожидаем увидеть некую константную ошибку, имеющую порядок 10^{-8}.
Полученный результат изобразим на графике (см. рис. 6)

Рисунок 6: Тест 2 версии 1 с округленном вычислением dt

Как мы видим результаты колеблются около порядка 10^{-8}. Более того, график начинается не из нуля, а со значения порядка 10^{-8}.
Можно сказать, что гипотеза доказана.

Гипотеза 2 - погрешность при умножения:

Аналогично с предыдущей гипотезой, так как dt мы только умножаем и других операций умножения в программе нет, получаем:

$$x_n = x_0 + nv_0 dt(1+e) + n^2 \frac{adt^2}{2}(1+2e)$$

$$x_n - x_{real} = e v_0 t + 2e \frac{at^2}{2}$$

$$E = \frac{e v_0 t + e at^2}{x_0 + v t + a t^2/2}$$

Проверка:
Попробуем проверить также как мы проверяли в проверке 2 предыдущего пункта.
Мы приведем к типу float результат умножения v на dt. Напомним, что во втором тесте остальные умножения отсутствуют (начальное положение скорость равны 0).
Внесем следующие изменения (а именно в процедуру Euler_for_structs):

```
float temp = v->x * dt;
r->x = r->x + temp + a.x * dt2;
```

Результаты измененной программы изобразим на графике (см. рис 7).

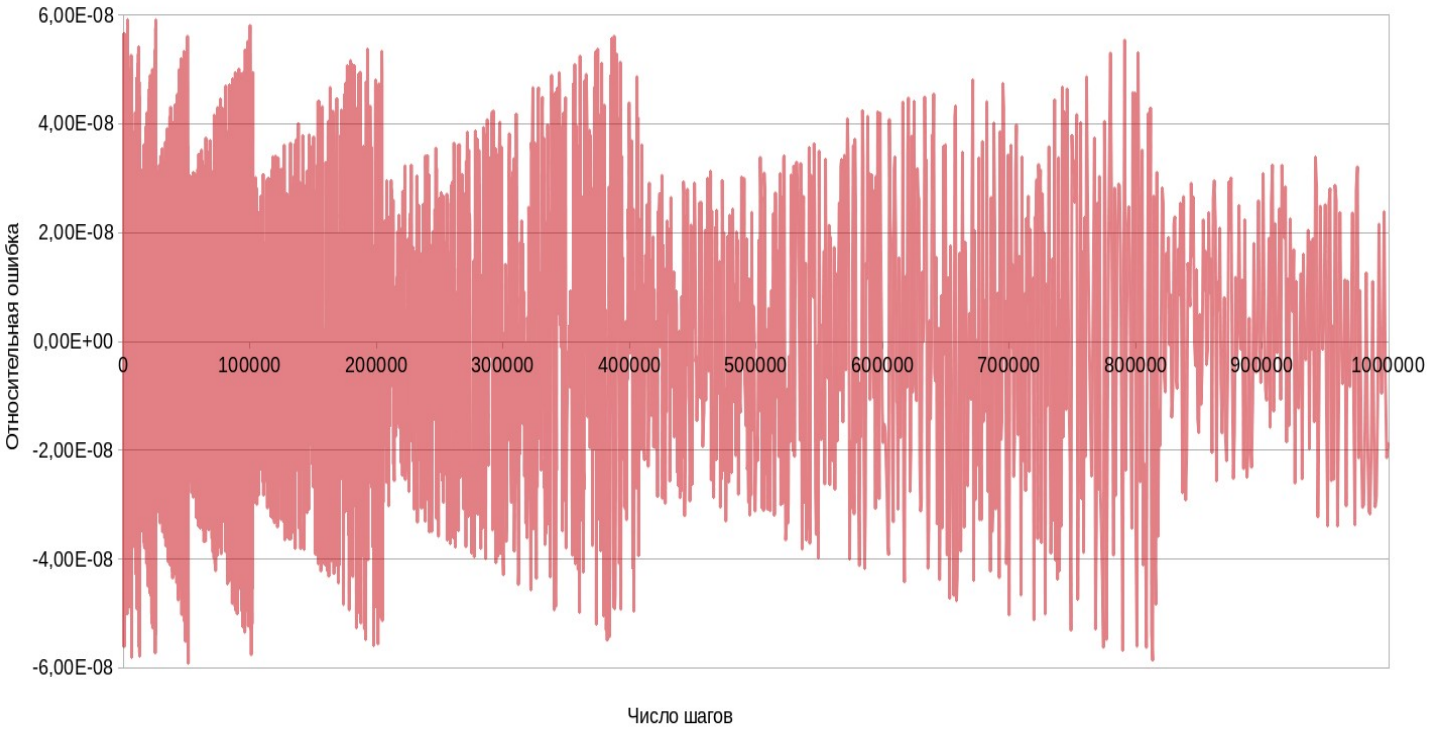

Рисунок 7: Тест 2 версии 1 с округленном умножением

Как мы видим результаты опять имеют порядок около 10^{-8}.
Можно ли считать, что гипотеза доказана.
Интересный факт:
Проверим также как и в первой гипотезе.
Для этого мы округлим умножение с помощью функции my_round, использованной в прошлой гипотезе.
Изменения в программе

```
r->x = r->x + my_round((v->x * dt), 12);
```

Напомню, что во втором тесте a = 0, v = 10. Поэтому относительная ошибка будет в 10 раз меньше, чем в предыдущем варианте (т. к. перед округлением мы умножаем dt на v = 10).
Полученные результаты программы с округленном умножением до 12 знака представим на графике (см. рис. 6):

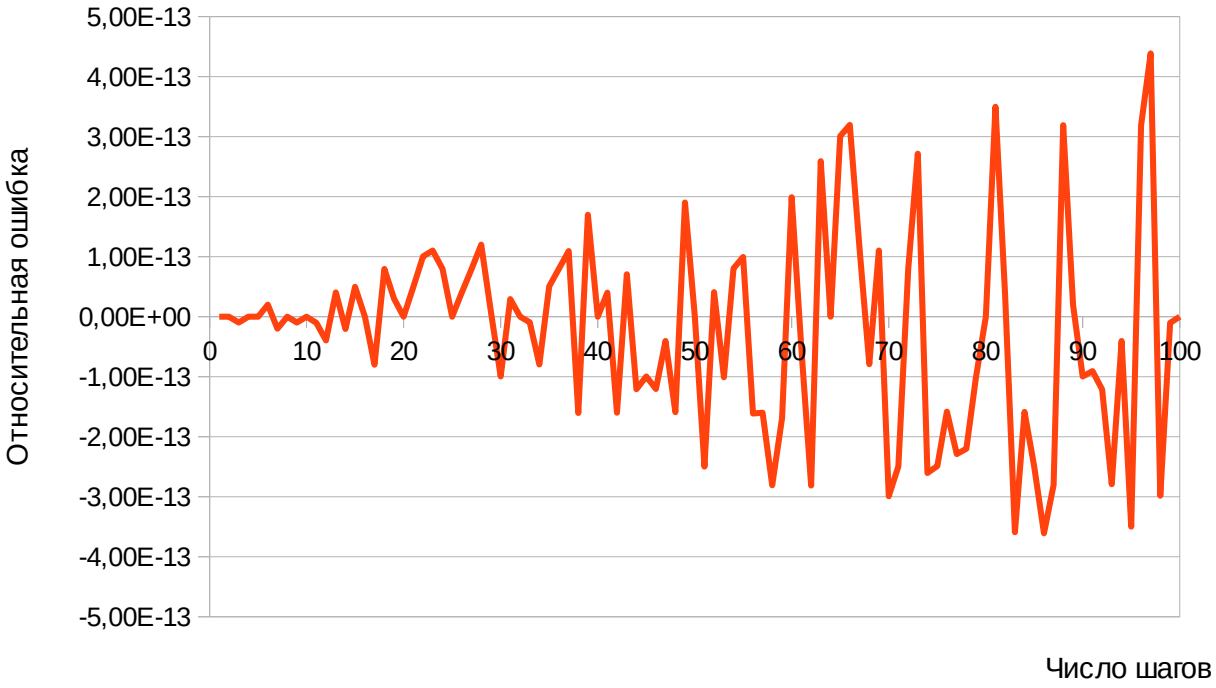

Рисунок 8: Тест 2 версии 1 с округленном умножением

Результаты имеют порядок 10^{-13}. Но опять растут линейно из нуля. Аналогичный случай, как в предыдущем параграфе.

Гипотеза 3 - погрешность при сложении:

Найдем погрешность метода, приняв, что операция сложения выполняется не точно.

Найдем погрешность при подсчете скорости:

$$v_1=(v_0+adt)(1+e)$$
$$v_2=(v_1+adt)(1+e)$$
$$v_2=v_0(1+e)^2+adt(1+e)^2+adt(1+e)$$

тогда , используя формулу для суммы арифметической прогрессии :

$$v_i=v_0(1+ie)+iadt\left(1+e\frac{i+1}{2}\right)$$

Найдем погрешность при вычислении координат:

$$x_1=((x_0+v_0dt)(1+e)+adt^2/2)(1+e)=x_0(1+2e)+v_0dt(1+2e)+adt^2/2(1+e)$$
$$x_2=((x_1+v_1dt)(1+e)+adt^2/2)(1+e)=x_1(1+2e)+v_1dt(1+2e)+adt^2/2(1+e)=$$
$$=x_0(1+4e)+v_0dt(1+4e)+v_1dt(1+2e)+adt^2/2(1+3e)+adt^2/2(1+e)$$

Тогда: $x_i=x_0(1+2ie)+v_0dt(1+2ie)+..+v_{(i-1)}dt(1+2e)+adt^2/2(1+e)+..+adt^2/2(1+(2i-1)e)$

Используя формулу для ар. прогрессии для преобразования суммы ускорений:

$$x_i=x_0(1+2ie)+v_0dt(1+2ie)+..+v_{(i-1)}dt(1+2e)+i\frac{a\,dt^2}{2}(1+ie)$$

Зная, как вычисляется скорость:

$$v_j = v_0(1+je) + j\,a\,dt\left(1+e\frac{j+1}{2}\right), \text{ где j = 0.. i - 1}$$

$$v_j \cdot dt\,(1+e)^{2j+2} = [v_0(1+je) + j\,a dt\,(1+e\cdot(\frac{j+1}{2}))]\cdot dt\,(1+e)^{2j+2} =$$

$$= v_0 \cdot dt\,(1+e(3j+2)) + \frac{j\cdot a\,dt^2}{2}(1+(2{,}5\,j+2{,}5)e)$$

С помощью формулы арифметической прогрессии просуммируем от j = 0 до j = i - 1:

$$\sum_{j=0}^{j=i-1} v_j \cdot dt\,(1+e)^{2j+2} = i\,v_0\,dt\left(1+\frac{3i+1}{2}e\right) + i(i-1)\frac{a\,dt^2}{2}\left(1+2{,}5\,e\frac{(i+1)}{2}\right)$$

Тогда:

$$x_n = x_0(1+2ne) + n\,v_0\,dt\left(1+e\frac{3n+1}{2}\right) + \frac{a\,dt^2\,n(n-1)}{2}\left(1+e\frac{2{,}5(n+1)}{2}\right) + n\frac{a dt^2}{2}(1+n\,e)$$

Немного упростим ускорение:

$$(n-1)\,n\frac{a\,dt^2}{2} + n\frac{a\,dt^2}{2} + e\,n(n-1)2{,}5\frac{(n+1)}{2}\frac{a\,dt^2}{2} + e\,n^2\frac{a\,dt^2}{2} =$$

$$= \frac{n^2\,a\,dt^2}{2}\left(1+e\frac{2{,}5(n+1)(n-1)}{2n}+e\right) = \frac{n^2\,a\,dt^2}{2}\left(1+e\frac{2{,}5\,n^2+2n-2{,}5}{2n}\right)$$

В итоге получаем:

$$dt = \frac{t}{n}$$

$$x_n = x_0(1+2n\,e) + v_0\,t\left(1+\frac{3n+1}{2}e\right) + \frac{a\,t^2}{2}\left(1+e\frac{2{,}5n^2+2n-2{,}5}{2n}\right)$$

$$x_{real} = x_0 + v_0\,t + \frac{a\,t^2}{2}$$

$$E = \frac{x_n - x_{real}}{x_{real}} = \frac{e\,x_0\,2n + e\,v_0\,t\frac{3n+1}{2} + e\frac{a\,t^2}{2}\frac{2{,}5n^2+2n-2{,}5}{2n}}{x_{real}}$$

Как мы видим погрешность накапливается линейно.

Проверим нашу формулу экспериментально.

Проверка 1:

Введем новый тест.

Тест 3 — равноускоренное линейное движение:

Возьмем следующие входные данные:

K = 0;

r_0 = (100 ; 0 ; 0)

v_0 = (10 ; 0 ; 0)

a = (10; 0 ; 0)

t = 10

n = 1000

x_{real} = 700

По выведенной выше формуле погрешность на N = 1000 шагах будет максимального порядка 10^{-14}.

Полученные результаты программы имеют именно такой порядок.

На всякий случай для проверки линейного характера проведем еще одну проверку.

Проверка 2:

Приведем пример линейного накопления погрешности (как это было в тесте 2, см рисунок 3) при a = 0:

$$x_1=(x_0+v_0\,dt)(1+e)$$

$$x_2=(x_1+v_1\,dt)(1+e)=x_0(1+2e)+v_0\,dt(1+2e)+v_1\,dt(1+e)$$

Рассуждая также как и в предыдущих выводах, используя формулу для суммы арифметической прогрессии и учитывая, что $v_0=v_1=..=v_n$, т.к. a = 0 получаем:

$$x_n=x_0(1+ne)+n\,v\,dt\,(1+e\frac{n+1}{2})$$

$$x_{real}=x_0+n\,v\,dt$$

$$E=\frac{e\,x_0\,n+e\,n\,v\,dt\,(n+1)/2}{x_0+n\,v\,dt}$$

Воспользуемся тестом 2ой первой версии. График представлен выше (см. рис. 3).

Для точной проверки в программу были внесены следующие изменения:

- введен новый формат вектора, теперь состоящий из полей long double
- для удобства использованы директивы условной компиляции

Вторая версия программы — переход к long double:

```
#ifndef K
#  define K   39.856e+13
#endif
#ifndef NMIN
#  define NMIN 01
#endif
#ifndef NMAX
#  define NMAX 10000000
#endif

struct vector {
    double x;
    double y;
    double z;};

struct precvector {
    long double x;
    long double y;
    long double z;};



void Accel_for_structs (struct precvector *r, struct vector *a);

void Euler_for_structs (struct precvector *r, struct vector *v,
double dt, int m);
```

```
int main () {
    ...

    for (n = NMIN; n <= NMAX; n += 1 + (n/500)+ (n/1000)+
(n/10000) ) {
   dt = t / n;
   r1 = r0;
   v1 = v0;
   Euler_for_structs(&r1, &v1, dt, n);
   printf("%d;%.14LG\n", n, (r1.x - real_x)/ real_x);
    }

    return 0;
}
```

Тестирование:

Используем тест 2 для проверки.

Ожидаемая точность должна отличаться в 1000 раз (улучшенная точность у long double).

Полученные результаты представим на графике (см. рис. 7):

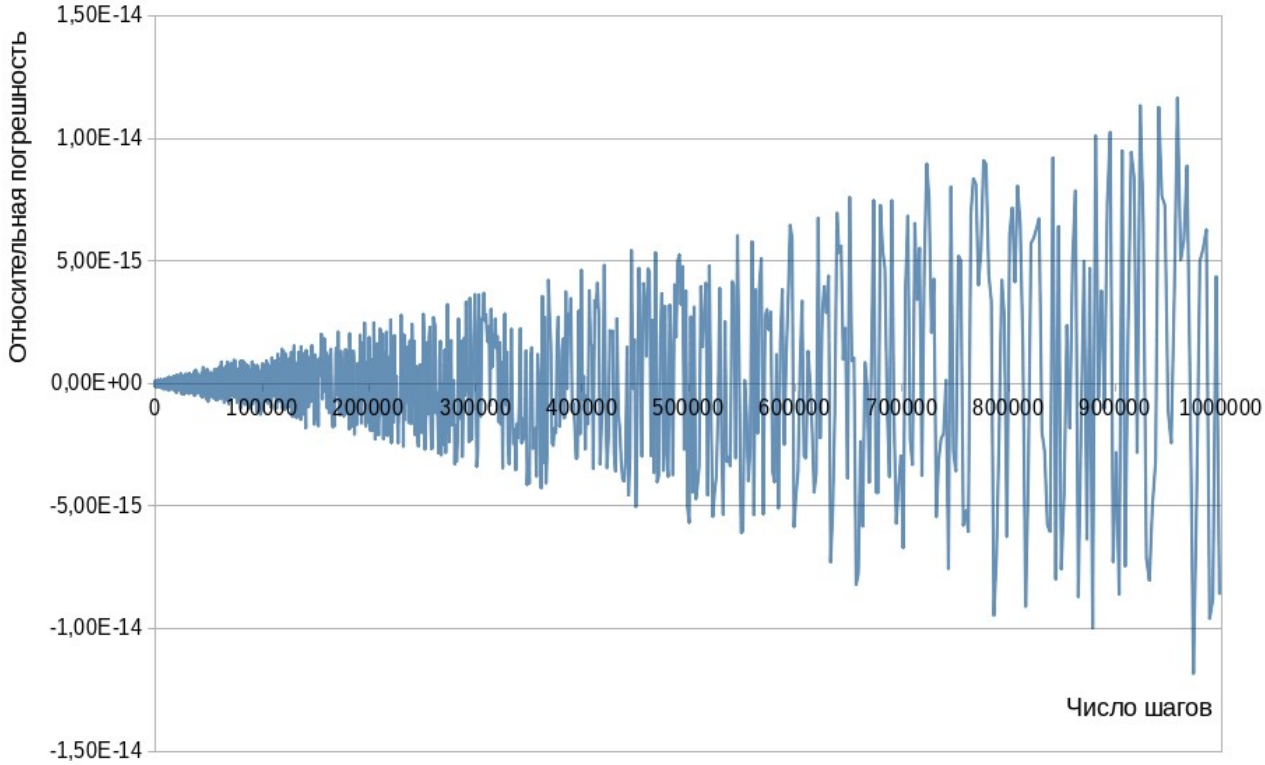

Рисунок 9: Тест 2 версии 1 с переходом к long double

Заметим, что на графике (рис. 4) при n = 1 000 000 относительная ошибка имеет порядок 10^{-14}. А на таком же графике предыдущей версии (см. рис. 7) при таком же n относительная ошибка имеет порядок 10^{-11}.

Точность увеличилась в 10^3 раз, как и ожидалось.
Гипотеза доказана.

Реализация метода Кехена:

Суммирование методом Кехана поможет избавится от линейного накопления погрешностей. А именно в формуле:

$$x_n = x_0(1+2n\,e)+v_0 t\left(1+\frac{3n+1}{2}e\right)+\frac{at^2}{2}\left(1+e\frac{2{,}5n^2+2n-2{,}5}{2n}\right)$$

Для этого произведем в программе следующие изменения:
- вернулись к double
- модернизирована Accel_for_structs: количество операций умножения для вычисления a — ускорения уменьшилось на 1.
- вычисление вектора радиуса и вектора ускорения теперь осуществляется через метод Кехана.

Третья версия программы:

```
/* Sum in long double compensated */
void sum_cmp(double a, double b, double *sum, double *t) {
    *sum = a + b;
    double b_ = *sum - a;
    double db = b - b_;
    *t = db;
    }

void Accel_for_structs (struct precvector *r, struct vector *a) {
    double r2 = r->x * r->x + r->y * r->y + r->z * r->z;
    if (r2 == 0) {
   a->x = 0;
   a->y = 0;
   a->z = 0;
    }
    else
    r2 = - K / (r2 * sqrt(r2));
    a->x = r->x * r2;
    a->y = r->y * r2;
    a->z = r->z * r2;
}

void Euler_for_structs (struct precvector *r, struct vector *v,
double dt, int m) {
    int i, j;
    double dvx, dvy, dvz;
    double mst_x = 0.0, mst_y = 0.0, mst_z = 0.0;
```

```
    double mst_vx = 0.0, mst_vy = 0.0, mst_vz = 0.0;
    struct vector a;
    for (i = 1; i <= m; i++) {
   /* Calculating accelerations */
   Accel_for_structs(r, &a);

   /* Summing radius-vectors */
   dvx = (v->x + a.x * dt / 2) * dt + mst_x;
   dvy = (v->y + a.y * dt / 2) * dt + mst_y;
   dvz = (v->z + a.z * dt / 2) * dt + mst_z;
   sum_cmp(r->x, dvx, &r->x, &mst_x);
   sum_cmp(r->y, dvy, &r->y, &mst_y);
   sum_cmp(r->z, dvz, &r->z, &mst_z);

   /* Summing velocities */
   a.x *=dt; a.y *= dt; a.z *= dt;
   a.x += mst_vx;
   a.y += mst_vy;
   a.z += mst_vz;
   sum_cmp(v->x, a.x, &v->x, &mst_vx);
   sum_cmp(v->y, a.y, &v->y, &mst_vy);
   sum_cmp(v->z, a.z, &v->z, &mst_vz);
    }
    /* Adding last mistakes if possible */
    r->x += mst_x;
    r->y += mst_y;
    r->z += mst_z;
    v->x += mst_vx;
    v->y += mst_vy;
    v->z += mst_vz;
}
```

Результаты:

Полученные результаты работы программы для теста 1 и теста 3 (из 6 параграфа) отобразим на графиках (см. рис. 8 и рис. 9 соответственно).

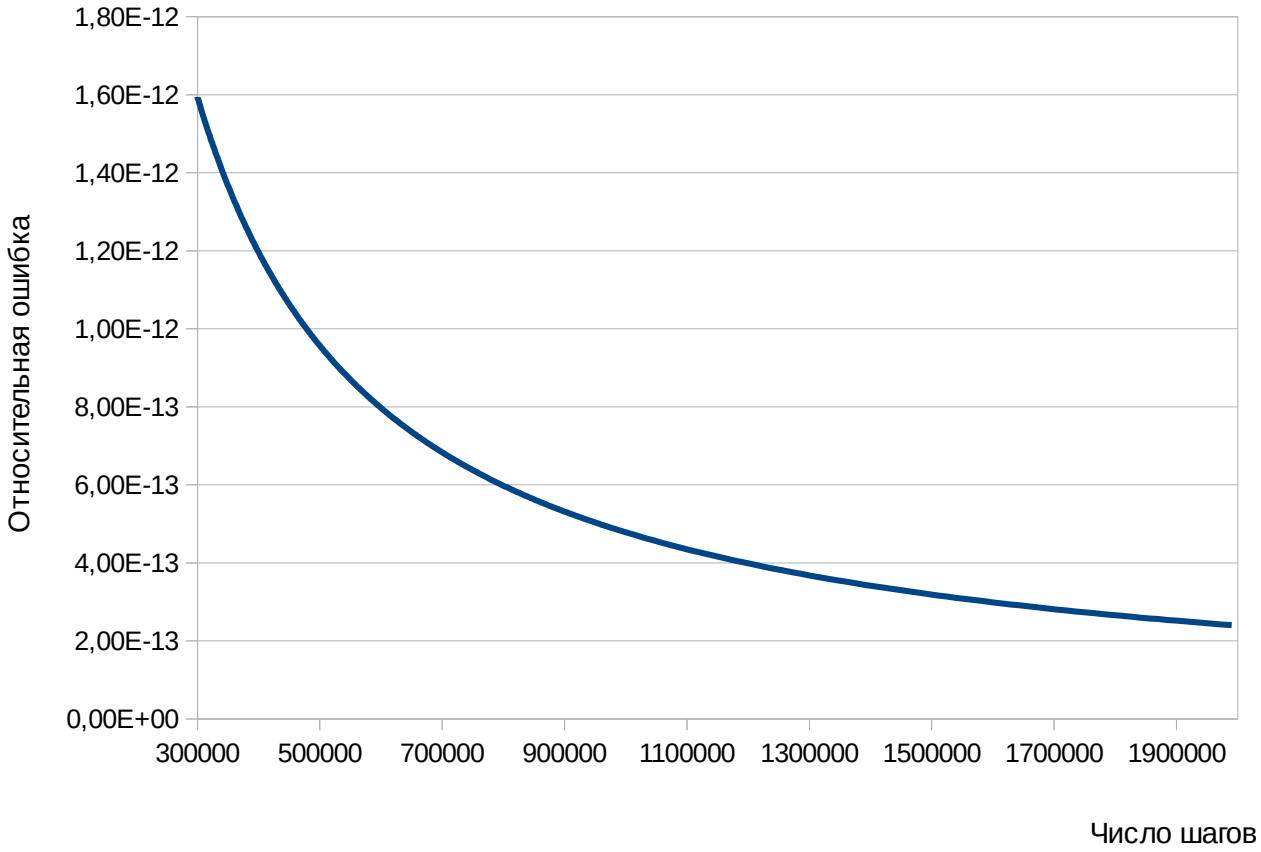

Рисунок 10: Тест 1 3 версии (с реализованным методом Кехена)

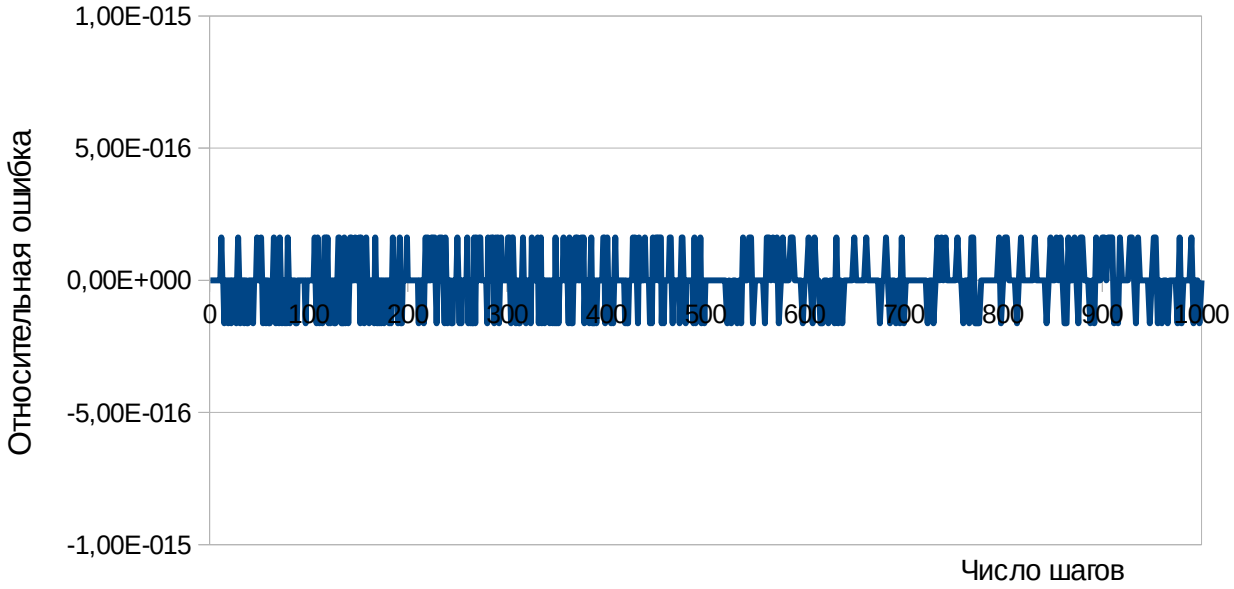

Рисунок 11: Тест 3 версии 3 (с реализованным методом Кехена)

Как мы видим, по рис. 9 алгоритм Кехена полностью убрал линейное накопление погрешности в данном тесте. Теперь при линейном движении погрешность метода колеблется в пределах машинной точности.
Также заметные улучшения в результатах теста 1 (движение по земной орбите): ошибка уменьшается с числом шагом, как и ожидалось (см. рис. 8)
Такие результаты можно считать удовлетворительными.