

НИУ «МЭИ»

Кафедра Радиотехнических систем

Расчетно-пояснительная записка к курсовому проекту по  
дисциплине «Аппаратура потребителей СРНС»

Выполнил: Капитонов А.И.

Группа: ЭР-15-17

Проверил: доц. Корогодин И.В.

Москва 2022

### **Этап 3. Моделирование в C++**

Требуется разработать на языке C/C++ функцию расчета положения спутника GPS на заданное время по шкале UTC, минимизируя время её исполнения и количество затрачиваемой оперативной памяти. Вызов функции не должен приводить к выбросу исключений или утечкам памяти при любом наборе входных данных.

Функции расчета положения спутника в Matlab относительно проста, т.к. доступны библиотеки линейной алгебры и решения уравнений. Но при разработке встраиваемого ПО приходится сохранять лицензионную частоту, минимизировать вычислительную нагрузку и затраты памяти. Требуется выполнить свою реализацию решения трансцендентного уравнения.

Программный модуль должен сопровождаться unit-тестами под check:

- Тесты функций решения уравнения Кеплера.
- Тест расчетного положения спутника в сравнении с Matlab с шагом 0.1 секунды.

Во время второго теста должно вычисляться и выводиться средняя длительность исполнения функции. Допускается использовать одни и те же эфемериды на весь рассматриваемый интервал.

Требуется провести проверку на утечки памяти.

### **Разработка алгоритма расчета положения спутника на языке C++**

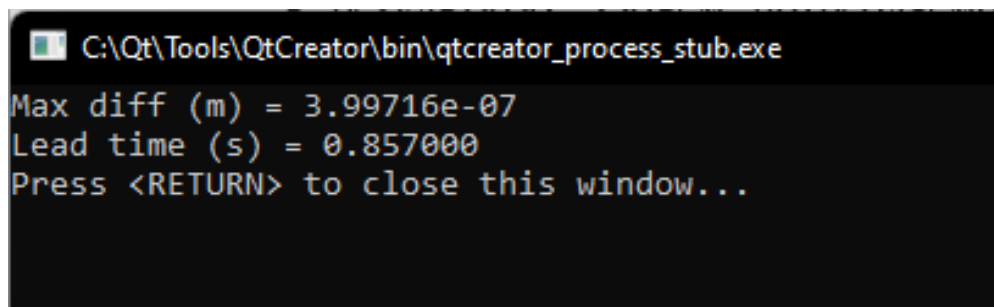
Функция реализована на языке программирования C/C++ в среде разработки QtCreator. Для простоты чтения кода программы текст был разделен на заголовочный файл `main.h` и исполнительный `main.cpp`. Функция, помимо вычислений координат, так же считывает координаты из файла, сравнивает их с рассчитанными, находит максимальную разницу, а также вычисляет общее время выполнения с помощью встроенной библиотеки `ctime`.

Для точного расчета, в ходе выполнения моделирования, была написана функция, выполняющая расчет уравнение Кеплера:

```
static double Kepler(double M){  
    double E = M;  
    double E_old = M + 1;  
    double epsilon = 1e-6;  
  
    while (abs(E - E_old) > epsilon) {  
        E_old = E;  
        E = M + Eph.e * sin(E);  
    }  
    return E;  
}
```

Рисунок 8 — Алгоритм решения уравнения Кеплера

После выполнения расчета получим результат моделирования данного алгоритма и время, затраченное на исполнение кода (рисунок 9).



A screenshot of a terminal window with a black background and white text. The title bar at the top reads "C:\Qt\Tools\QtCreator\bin\qtcreator\_process\_stub.exe". The output text in the terminal is: "Max diff (m) = 3.99716e-07", "Lead time (s) = 0.857000", and "Press <RETURN> to close this window...".

Рисунок 9 — Результат работы программы

Анализ утечек памяти был проведен в ос Ubuntu с помощью утилиты valgrind. Результат анализа представлен на рисунке 10.

```
==1977== LEAK SUMMARY:  
==1977==    definitely lost: 0 bytes in 0 blocks  
==1977==    indirectly lost: 0 bytes in 0 blocks  
==1977==    possibly lost: 0 bytes in 0 blocks  
==1977==    still reachable: 72,704 bytes in 1 blocks  
==1977==    suppressed: 0 bytes in 0 blocks
```

Рисунок 10 — Анализ утечки памяти

## Вывод:

На третьем этапе курсового проекта была написана программа на языке C/C++ так как этот язык является низкоуровневым и идеально подходит для разработки встраиваемого ПО. Серьезных расхождений между расчетами в данной программе и расчетами в Matlab не обнаружилось. Так же был проведен тест, показывающий отсутствие утечек памяти.

## Приложение к этапу 3

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <ctime>

#include "main.h"

using namespace std;

Ephemeris Eph;

static int GetTime() {
    begin_time = (24 * 0 + 2) * 60 * 60;
    end_time = (24 * 1 + 2) * 60 * 60;
    return end_time - begin_time;
}

static double Kepler(double M) {
    double E = M;
    double E_old = M + 1;
    double epsilon = 1e-6;

    while (abs(E - E_old) > epsilon) {
        E_old = E;
        E = M + Eph.e * sin(E);
    }
    return E;
}

int main()
{
    srand(time(0));

    //Длина временного промежутка
    int t_arr = GetTime();

    // Среднее движение
    double n0 = sqrt(mu / pow(Eph.A, 3));
    double n = n0 + Eph.Dn;

    // Координаты
    double** coord = new double*[3];
    for (int i = 0; i < 3; i++)
        coord[i] = new double[t_arr];

    // Координаты из матлаба
    double** coordMat = new double*[3];
```

```

for (int i = 0; i < 3; i++)
    coordMat[i] = new double[t_arr];

for (int k = 0; k < t_arr; k++) {
    // Время
    double t = begin_time + k - Eph.toe;

    if (t > 302400)
        t = t - 604800;
    if (t < -302400)
        t = t + 604800;

    // Средняя аномалия
    double M = Eph.M0 + n * t;

    double E = Kepler(M);

    // Истинная аномалия
    double nu = atan2(sqrt(1 - pow(Eph.e, 2)) * sin(E), cos(E) - Eph.e);

    // Коэффициент коррекции
    double cos_correction = cos(2 * (Eph.omega + nu));
    double sin_correction = sin(2 * (Eph.omega + nu));

    // Аргумент широты
    double u = Eph.omega + nu + Eph.Cuc * cos_correction + Eph.Cus *
sin_correction;

    // Радиус
    double r = Eph.A * (1 - Eph.e * cos(E)) + Eph.Crc * cos_correction +
Eph.Crs * sin_correction;

    // Наклон
    double i = Eph.i0 + Eph.iDot * t + Eph.Cic * cos_correction + Eph.Cis
* sin_correction;

    // Долгота восходящего угла
    double lambda = Eph.Omega0 + (Eph.OmegaDot - omega_e) * t - omega_e *
Eph.toe;

    // Положение на орбите
    double x = r * cos(u);
    double y = r * sin(u);

    // Координаты
    double Xk = x * cos(lambda) - y * cos(i) * sin(lambda);
    double Yk = x * sin(lambda) + y * cos(i) * cos(lambda);
    double Zk = y * sin(i);

    coord[0][k] = Xk;
    coord[1][k] = Yk;
    coord[2][k] = Zk;
}

// Чтение координат из матлаба
ifstream file("out_m.txt");

if (file.is_open()){
    for (int k = 0; k < t_arr; k++)
        file >> coordMat[0][k] >> coordMat[1][k] >> coordMat[2][k];
    file.close();
}
else
    printf("Can't open file!\n");

```

```

// Сравнение с матлабом
double max_del = 0;
for (int i = 0; i < 3; i++) {
    for (int k = 0; k < t_arr; k++) {
        if (abs(coord[0][k] - coordMat[0][k]) > max_del)
            max_del = abs(coord[0][k] - coordMat[0][k]);
    }
}

// Очищение памяти
delete[] coord;
coord = nullptr;
delete[] coordMat;
coordMat = nullptr;

printf("Max diff (m) = %g\n", max_del);
printf("Lead time (s) = %f\n", clock()/1000.0);
}

```

```

#ifdef MAIN_H
#define MAIN_H
#include <math.h>

typedef struct _Ephemeris{
    int toe = 93600;
    double Crs = 1.219062e+02;
    double Dn = 1.301601e-09;
    double M0 = 1.744108e+01;
    double Cuc = 6.347895e-06;
    double e = 8.970004e-03;
    double Cus = 7.888302e-06;
    double sqrtA = 5.153688e+03;
    double Cic = -5.587935e-08;
    double Omega0 = -1.640395e+02;
    double Cis = -9.685755e-08;
    double i0 = 5.608838e+01;
    double Crc = 2.403438e+02;
    double omega = 1.150381e+02;
    double OmegaDot = -4.577919e-07;
    double iDot = 1.029321e-08;
    double A = pow(sqrtA,2);
} Ephemeris;

extern Ephemeris Eph;

// Значения констант
double mu = 3.986004418e14;
double omega_e = 7.2921151467e-5;

int begin_time;
int end_time;

#endif // MAIN_H

```