Домашнее задание по предмету "Архитектура компьютеров"

Выполнил студент ИУ9-12
Белогуров Алексей

2014г.

Оглавление

Этап 1: **Постановка задачи**

Задача: Разработать подпрограмму численного моделирования упругих колебаний системы тел, связанных упругими стержнями. Задаётся число тел, для каждого тела его масса m_i, координаты в пространстве $\bar{r}_i$ и коэффициенты упругости стержней $K_{ij} = K_{ji}$, связывающих его с другими телами. В исходном состоянии стержни не нагружены (находятся в свободном состоянии). В момент времени T_0 на одно или несколько тел действует начальный импульс, мгновенно придающий этим телам скорости $\bar{v}_i$. Силы упругости действуют строго вдоль оси стержня, величина силы, действующей со стороны тела j на тело i определяется коэффициентом упругости K_{ij} и величиной деформации стержня:
$\bar{F}_{ij} = \frac{\bar{L}_{ij}}{|\bar{L}_{ij}|} \cdot K_{ij} \cdot (|\bar{L}_{ij}| - L_{fij}), \bar{L}_{ij} = \bar{r}_j - \bar{r}_i$, а $L_{fij} = |\bar{L}_{fij}|$ - длина стержня в свободном состоянии(в момент времени T_0). Ускорение тела i равно $\bar{a}_i = \sum_{j=1..N, j\neq i} \frac{\bar{F}_{ij}}{m_i}$.

Для интегрирования дифференциальных уравнений применять:
скоростной метод Верле (velocity Verlet Scheme):
В скоростном методе Верле предполагается, что ускорение тела $\bar{a}_t$ в момент времени t зависит только от положения тела в пространстве: $\bar{a}_t = f(\bar{r}_t)$, тогда положение и скорость тела в следующий момент времени $t + \Delta t$ могут быть определены как

$$\begin{gathered}
\bar{r}_{t+\Delta t} = \bar{r}_t + \bar{v}_t \cdot \Delta t + \tfrac{1}{2} \cdot \bar{a}_t \cdot \Delta t^2, \\
\bar{v}_{t+\frac{1}{2}\Delta t} = \bar{v}_t + \tfrac{1}{2} \cdot \bar{a}_t \cdot \Delta t, \\
\bar{a}_{t+\Delta t} = f(\bar{r}_{t+\Delta t}), \\
\bar{v}_{t+\Delta t} = \bar{v}_{t+\frac{1}{2}\Delta t} + \tfrac{1}{2} \cdot \bar{a}_{t+\Delta t} \cdot \Delta t,
\end{gathered}$$

где $\bar{r}, \bar{v}$ и $\bar{a}$ - радиус-вектор, скорость и ускорение в моменты времени t и $t + \Delta t$ с шагом Δt. Примечание: для уменьшения погрешностей скорость определяется в ещё одной промежуточной точке $t + \frac{1}{2}\Delta t$.

Дополнительные указания:
a) Использовать тип данных float;
b) Программа должна быть универсальной, корректно работающей для свободно изменяемых начальных условий;
c) Начальные условия (параметры функций, сами функции и т.п.) ограничены областями определения используемых математических функций и представимыми в заданном формате величинами;
d) Во время выполнения работы можно и нужно варьировать начальные условия;
e) Для обнаружения ошибок имеет смысл использовать такие параметры тестов, чтобы результат мог быть легко проверен аналитически; хотя в общем виде надо учитывать, что численное решение применяется тогда, когда аналитическое невозможно или затруднительно (например, очень большое число объектов) и при разработке программы надо предполагать невозможность аналитического вычисления результата.

Этап 2: **Разработка работоспособной программы**

Рассмотрим наивную реализацию программы *(Листинг 1)*

Листинг 1. Наивная реализация

```
#include <stdio.h>
#include <math.h> /* определения констант и мат. функций */
#include <locale.h> /* прототип setlocale и констант LC_... */

float Power(float coord_i, float coord_j, float k, float len_ij, float len_0ij)
/* Функция для вычисление силы F для ускорения с помощью координат i, j, коэффициента
   k, длины в данный момент времени(len_ij) и в начальный момент времени(len_0ij) */
{
     return ((coord_j-coord_i)*k/len_ij*(len_ij - len_0ij));
}

float LenVector(float r[][3], int i, int j)
/* Функция для вычисления длины стержня между телами i и j */
{
     float len;
     len = pow(r[i][0] - r[j][0],2) + pow(r[i][1] - r[j][1],2) + pow(r[i][2] -
                                                                    r[j][2],2);
     return sqrt(len);
}

float LenCoord(float coord_i, float coord_j)
/* Функция для вычисления разности координат i и j */
{
     return coord_i - coord_j;
}

int main(int argc, char **argv)
{
     setlocale( LC_ALL, "" );
     int n; /* количество тел в системе */
     scanf("%d\n", &n);
     float m[n], /* массив масс */
           r[n][3], /* двумерный массив координат */
           k[n][n], /* двумерный массив коэф. упругости между телами */
           v[n][3], /* двумерный массив проекций скоростей тел на оси x, y, z */
           a[n][n]; /* двумерный массив проекций ускорения тел на оси x, y, z */
     int i, j, q, p;
     for(i = 0; i < n; i++)
          scanf("%e %e %e %e %e %e %e %e %e %e %e", &m[i], &r[i][0], &r[i][1],
    &r[i][2], &k[i][1], &k[i][1], &k[i][2], &k[i][3], &v[i][0], &v[i][1], &v[i][2]);
    /* последовательное считывание значений координат, коэф. упругости, проекций
    скорости для тела i */

     float len_t0[n][n]; /* двумерный массив длин стержней в проекциях на оси в момент
                                                                    времени T0 */
     for(i = 0; i < n; i++) /* вычисление этих длин для последующего вычисления силы */
          for(q = 0; q < n; q++) {
               len_t0[i][q] = LenVector(r, i, q);
     }
```

```
 /* ВЫЧИСЛЕНИЕ НАЧАЛЬНОГО УСКОРЕНИЯ ДЛЯ ТЕЛА i */
 float power, at;
 for(i = 0; i < n; i++) {
      /* printf(" Ускорение в T0 для тела %d:\n", i); */
      for(j = 0; j < 3; j++) { /* j отвечает за ось; j = 0 - ось X,  j = 1 - ось Y,
                                                                    j = 2 - ось Z */
           at = 0;
           for(q = 0; q < n; q++) { /* q отвечает за второе тело */
                if (q == i)
                     continue;
                power = Power(r[i][j], r[q][j], k[i][q], LenVector(r, i, q),
                                                                    len_t0[i][q]);
                at += power/m[i];
           }
           /* printf(" %.15f, ", at); */
           a[i][j] = at; /* ускорение тела i на ось j */
      }
      /*printf("\n"); */
 }

 /* ГЛАВНАЯ ЧАСТЬ ПРОГРАММЫ, В КОТОРОЙ ПРИ КАЖДОЙ ИТЕРАЦИИ ИСПОЛЬЗУЮТСЯ ОСНОВНЫЕ
ФОРМУЛЫ ДЛЯ ВЫЧИСЛЕНИЯ КООРДИНАТ, СКОРОСТИ И УСКОРЕНИЯ, ПРИВЕДЕННЫЕ В УСЛОВИИ
ЗАДАЧИ */
 float dt, T;
 scanf("%e", &dt); /* считывание времени dt */
 T = dt;
 /* printf("dt - %.15f\n", dt); */
 int step;
 scanf("%d", &step); /* считывание необходимого количества шагов выполнения данного
                                                                        цикла */
 for(p = 0; p < step; T += dt, p++) {
      printf("\n Шаг № %d\n", p);

      /* ВЫЧИСЛЕНИЕ НОВЫХ КООРДИНАТ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt */
      for(i = 0; i < n; i++) {
           printf(" Координаты тела %d:\n", i);
           for(j = 0; j < 3; j++) {
                r[i][j] += v[i][j]*dt + a[i][j]*dt*dt/2;
                printf("%.15f, ", r[i][j]);
           }
           printf("\n");
      }

      /* ВЫЧИСЛЕНИЕ НОВОЙ СКОРОСТИ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt/2 */
      for(i = 0; i < n; i++) {
           /* printf(" Скорость тела %d:\n", i); */
           for(j = 0; j < 3; j++) {
                v[i][j] += a[i][j]*dt/2;
                /* printf("%.15f, ", v[i][j]); */
           }
           /* printf("\n"); */
      }

      /* ВЫЧИСЛЕНИЕ НОВОГО УСКОРЕНИЯ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt */
      for(i = 0; i < n; i++) {
```

```
                /* printf(" Ускорение тела %d:\n", i); */
                for(j = 0; j < 3; j++) {
                    at = 0;
                    for(q = 0; q < n; q++) {
                        if (q == i)
                            continue;
                        power = Power(r[i][j], r[q][j], k[i][q], LenVector(r, i, q),
                                                                        len_t0[i][q]);
                        at += power/m[i];
                    }
                    /* printf("%.15f, ", at); */
                    a[i][j] = at;
                }
                /* printf("\n"); */
            }

            /* ВЫЧИСЛЕНИЕ НОВОЙ СКОРОСТИ В МОМЕНТ ВРЕМЕНИ T+dt */
            for(i = 0; i < n; i++) {
                /* printf(" Скорость тела %d:\n", i); */
                for(j = 0; j < 3; j++) {
                    v[i][j] += a[i][j]*dt/2;
                    /* printf("%.15f, ", v[i][j]); */
                }
                /* printf("\n"); */
            }
        }

        printf("%f\n", T);
        return 0;
}
```

В результате работы программы на экран при каждой итерации главного цикла выводится номер № шага и последовательно координаты каждого тела на ось x, y и z, представленных в виде числа с плавающец точкой, в конце вывода можно наблюдать общий период времени движения тел, связанных упругими стержнями(рис. 1). Для удобоности проверки программы можно раскомментировать построчный вывод проекций ускорения и скорости всех тел.

```
Шаг № 148
Координаты тела 0:
-0,171952843666077 -1,234030485153198 21,036315917968750
Координаты тела 1:
-3,687892675399780 -5,848543643951416 12,283394813537598
Координаты тела 2:
-15,596325874328613 7,753028392791748 14,416337966918945
Координаты тела 3:
-18,229661941528320 4,967362880706787 3,907850503921509

Шаг № 149
Координаты тела 0:
-0,270349919795990 -1,253031611442566 21,029188156127930
Координаты тела 1:
-3,750571250915527 -5,766663074493408 12,312252044677734
Координаты тела 2:
-15,536535263061523 7,665657520294189 14,618625640869141
Координаты тела 3:
-18,290397644042969 4,930013179779053 4,001809597015381

Период времени движения тел: 15,000000
```

Рис. 1: Пример работы программы с данными, взятые из условия задачи.

Этап 3: **Проверка работы программы в случае с пружинным маятником**

Перед тем, как начать искать ошибки в реализации нашей программы, необходимо проверить ее работоспособность. Для этого мы можем рассмотреть задачу из школьного курса физики - свободные колебания грузика на пружине. Так как в нашей программе не учитывается использование ускорения свободного падения, то логично предположить, что наши колебания будут гармоническими, не затухающими.

Гармонические колебания — колебания, при которых физическая величина изменяется с течением времени по синусоидальному или косинусоидальному закону(рис. 2). Период колебаний T и амплитуда колебаний A являются постоянными величинами в данной задаче.

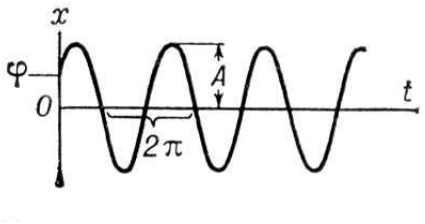

Рис. 2: График гармонических колебаний.

Для реализации свободных гармонических колебаний нам потребуется взять 2 тела, масса одного из которых будет намного больше массы другого, таким образом мы решим проблему движения обоих тел. В данных условиях находится в покое будет только тело большей массы, к нему мы и привяжем пружину со вторым телом более меньшей массы, которое будет исполнять роль грузика на пружине. Также чтобы вывести систему тел из положения равновесия необходимо будет придать некую скорость второму телу.
Для удобства проверки возьмем следующие данные:

$$m_1 = 10^9,\ m_2 = 1,$$
$$x_1 = 0,\ y_1 = 1000,\ z_1 = 0,$$
$$x_2 = 0,\ y_2 = 500,\ z_2 = 0,$$
$$v_{1x} = 0,\ v_{1y} = 0,\ v_{1z} = 0,$$
$$v_{2x} = 0,\ v_{2y} = 20,\ v_{2z} = 0,$$
$$k = 1,\ \Delta t = 0.1.$$

Для того, чтобы правильнее сравнить результаты нашей работы с предполагаемыми, используем только ось y, вдоль которой будут наблюдаться колебания(рис. 3).

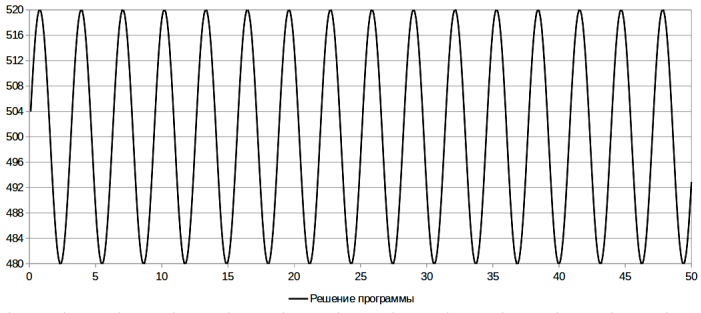

Рис. 3: График работы программы для свободных колебаний.

Как можно наблюдать, наш график(рис. 3) очень схож с графиком гармонических колебаний(рис. 2), в котором координата y изменяется с течением времени, а период колебаний T и амплитуда A являются постоянными величинами. Следовательно, наше предположение о том, что эти колебания являются гармоническими, не затухающими - верно. Также можно проверить верность вычисления амплитуды по следущей формуле:

$$A = v\sqrt{\frac{m}{k}} \tag{1}$$

$$A = 20[\tfrac{m}{s}] \cdot \sqrt{\frac{1[kg]}{1[\frac{N}{m}]}} = 20[m]$$

Как видно из нашего графика(рис. 3), амплитуда также равна 20. Чтобы удостовериться в правильности работы программы с различными данными, необходимо сравнивать получившиеся результаты с некоторым эталонным решением, а точнее с графиком эталонного решения. Таким решением может послужить уравнение свободных колебаний:

$$x = A \cdot cos(\omega \cdot t + \varphi_0) \tag{2}$$

Где A - амплитуда, φ_0 - начальная фаза, в нашем случае она будет равна $\frac{3\pi}{2}$, t возьмем равное 0.2, ω - собственная частота колебаний, которая может быть представлена как:

$$\omega = \frac{2\pi}{T} = \sqrt{\frac{k}{m}} \tag{3}$$

Таким образом, подставив уравнения амплитуды (1) и собственной частоты колебаний (3) в уравнение свободных колебаний (2) получим уравнение (4), в котором координата x зависит только от промежутка времени t, жёсткости пружины k, массы тела m и его скорости v, все эти величины нам известны по условию задачи.

$$x = v\sqrt{\frac{m}{k}} \cdot cos(\sqrt{\frac{k}{m}} \cdot t + \varphi_0) \tag{4}$$

Итак, сравним наши графики при разных Δt((рис. 4), (рис. 5), (рис. 6)), далее график программы - черный цвет, график эталонного решения - бирюзовый цвет:

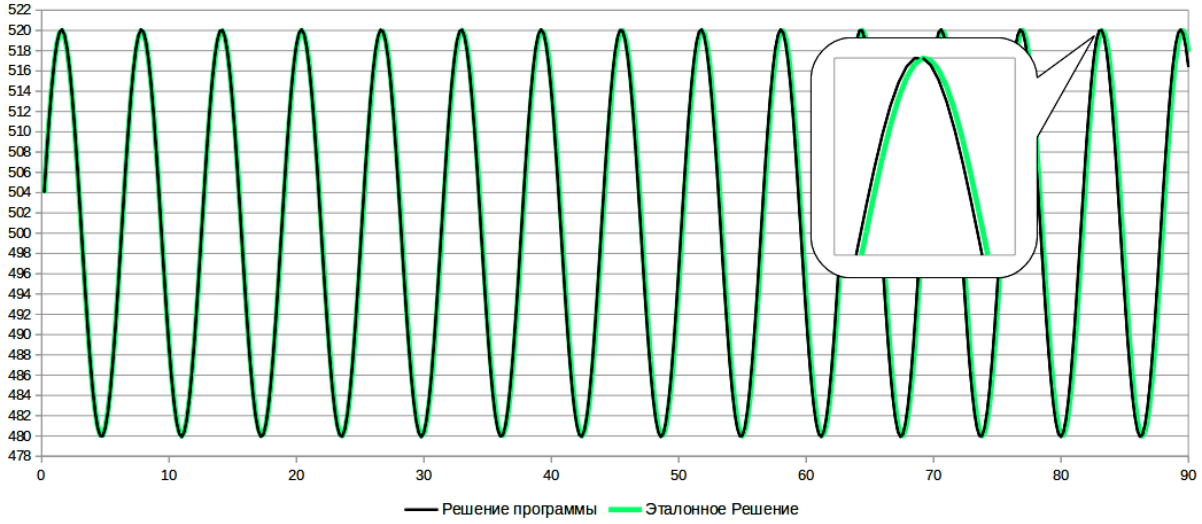

Рис. 4: Сравнение графиков колебаний при $\Delta t = 0.2$.

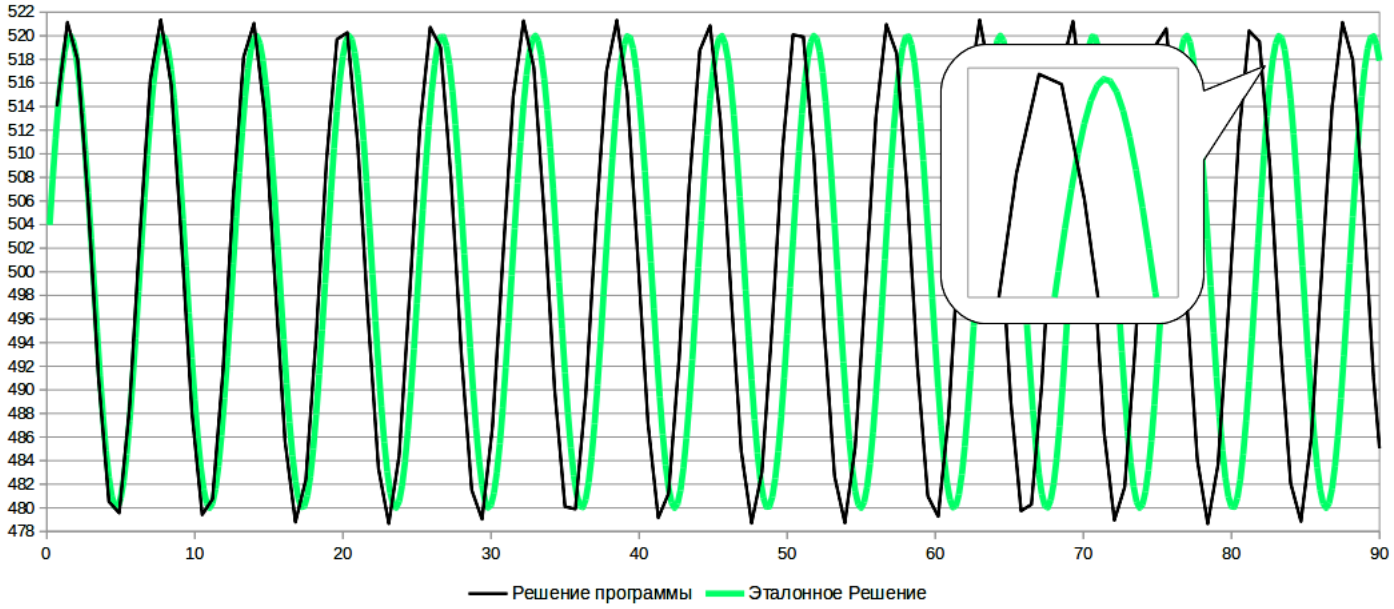

Рис. 5: Сравнение графиков колебаний при $\Delta t = 0.7$.

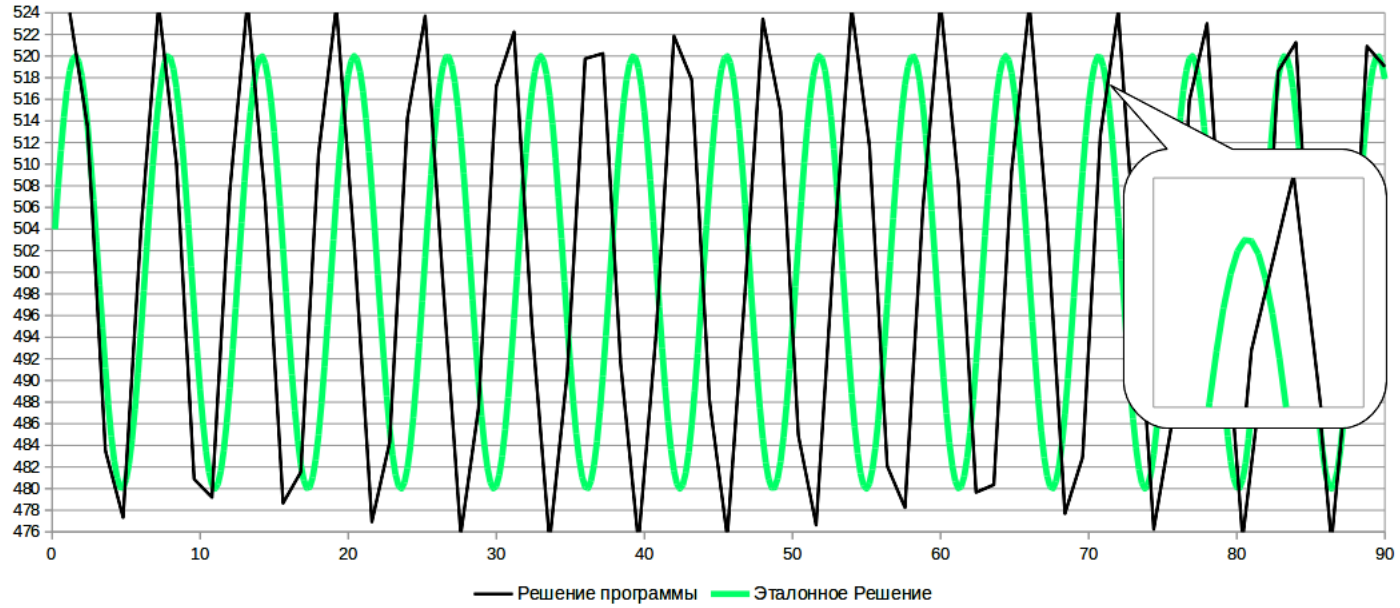

Рис. 6: Сравнение графиков колебаний при $\Delta t = 1.2$.

Получив графики колебаний при разных значениях Δt можно определенно сказать, что увеличение промежутка времени отрицательно сказывается на вычислениях программы, то есть при малом $\Delta t = 0.2$(рис. 4), график функции программы почти полностью покрывает график эталонного решения. При $\Delta t = 0.7$(рис. 5) видно, как амплитуда начинает превосходить амплитуду, найденную нами раннее(1). А при $\Delta t = 1.2$(рис. 6) видно уже очень грубое расхождение с графиком эталонного решения. Нетрудно предположить, что при больших Δt ситуация усложнится.

Это объясняется тем, что в точках, где тело достигает своего амплитудного значения, за-

висимость координаты от времени получается квадратичная, так как ускорение в этой точке максимально, а скорость стремится к 0. Следовательно, при большом Δt, грузику будет придаваться настолько большое ускорение, что при следующем шаге его реальная координата будет просто "улетать" за пределы амплитудных значений, что непозволительно. Аналогично можно сказать про точку положения равновесия, где зависимость уже будет линейной, так как скорость будет максимальна, а ускорение будет стремится 0. Также можно заметить, что при увеличении Δt период колебаний программы начинает не совпадать с периодом эталонного решения, это можно объяснить тем, что именно в тех точках, где координата превышает свое амплитудное значение, коэффициент жёсткости k также будет немного другим, следовательно при измении k будет изменяться и период колебаний T (6).

Чтобы избежать всего вышесказанного, необходимо в начале программы ввести ограничение на ввод значения Δt. Для этого Δt должно быть таким, чтобы значение координаты тела считалось такое число раз за один период, которое обеспечивало плавный график без сильных рывков. Для определенности, возьмем эту цифру равную 30:

$$\frac{T}{\Delta t} \geq 30 \tag{5}$$

$$T = 2\pi\sqrt{\frac{m}{k}} \tag{6}$$

Для наших данных получим $\Delta t \leq 0.21$, что очень схоже с временем, взятым на рис. 4. То есть программа должна вычислять по формуле (5) Δt и сверять его с тем, которое вводит пользователь. Если введенное Δt превышает заданное, программа выводит предупреждение о том, что последующие вычисления могут сильно отличаться от правильных. Внесенные изменения будут выглядеть следующем образом *(Листинг 2)*

Листинг 2. Ограничение на ввод времени

```
float dt, T, test_dt, pi = 3.14159265358;
test_dt = 2*pi*sqrt(m[1]/k[0][1])/30;
printf("Введите dt, которое не должно превышать следующего значения - %.6f\n",
                                                                    test_dt);
scanf("%e", &dt); /* считывание времени dt */
printf("dt - %.6f\n", dt);
     printf("Введенное dt превышает допустимое. Полученные ответы могут
                                          сильно отличаться от правильных.\n");
```

Поменяем входные данные на следующие:

$$\begin{gathered} m_1 = 10^9,\ m_2 = 1, \\ x_1 = 0,\ y_1 = 10^7,\ z_1 = 0, \\ x_2 = 0,\ y_2 = 5 \cdot 10^6,\ z_2 = 0, \\ v_{1x} = 0,\ v_{1y} = 0,\ v_{1z} = 0, \\ v_{2x} = 0,\ v_{2y} = 200,\ v_{2z} = 0, \\ k = 1,\ \Delta t = 0.1. \end{gathered}$$

И также сравним поведение получившегося графика с эталонным ((рис. 7), (рис. 8)):

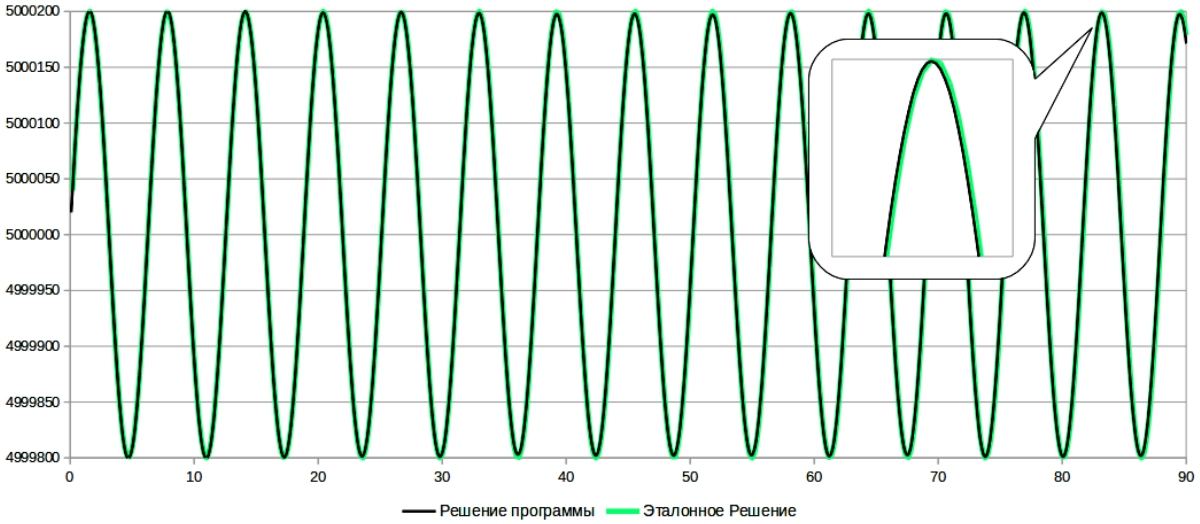

Рис. 7: Сравнение графиков колебаний при $\Delta t = 0.1$ в промежутке от 1...90.

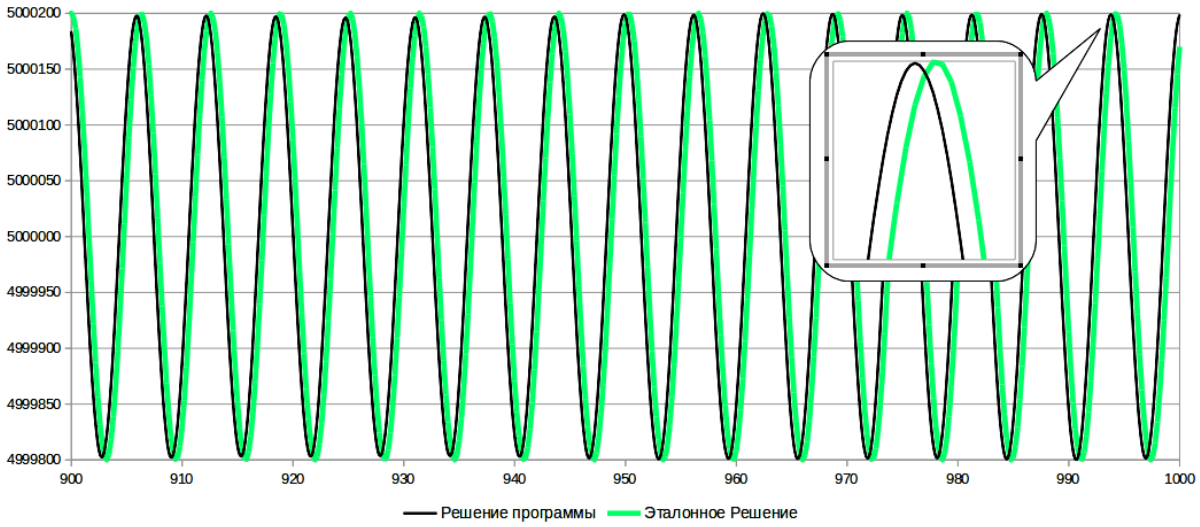

Рис. 8: Сравнение графиков колебаний при $\Delta t = 0.1$ в промежутке от 900...100.

Так как графики схожи друг другу, можно дальше повторять наши вычисления, еще раз увеличим показания координат:

$$m_1 = 10^9,\ m_2 = 1,$$
$$x_1 = 0,\ y_1 = 10^9,\ z_1 = 0,$$
$$x_2 = 0,\ y_2 = 5 \cdot 10^8,\ z_2 = 0,$$
$$v_{1x} = 0,\ v_{1y} = 0,\ v_{1z} = 0,$$
$$v_{2x} = 0,\ v_{2y} = 200,\ v_{2z} = 0,$$
$$k = 1,\ \Delta t = 0.1.$$

На графике(рис. 9) четко видно, что координата перестала быть плавной и тем более хотя бы совпадать с графиком эталонного решения, а это значит то, что в нашей программе производятся арифметичские операции с числами разных порядков, то есть числа больших порядков при сложении "съедают дробные части" чисел более малого порядка. Чтобы измежать данной ошибки мы применим метод суммирования Кохена.

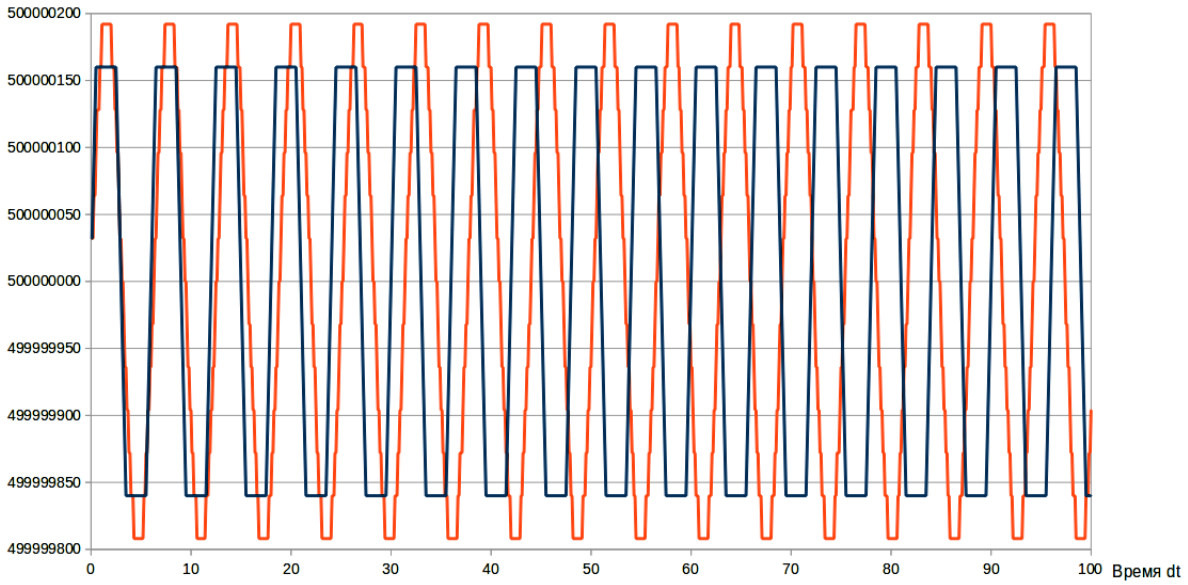

Рис. 9: Сравнение графиков колебаний при $\Delta t = 0.1$.

Метод суммирования Кохена - это метод, в котором для нахождения суммы большого множества чисел производится коррекция промежуточной суммы на протяжении всей работы алгоритма. Каждый член вначале корректируется согласно ошибке, накопившаяся на предыдущих этапах. Затем новая сумма вычисляется путем суммирования скорректированного члена и промежуточной суммы. После этого вычисляется поправочный член как разница между изменением в сумме и прибавленного скорректированного члена.

Так как суммирование у нас идет не только во время вычисления координаты, а также при вычислении скорости и ускорения на каждом шаге, то логично применить метод Кохена ко всем этим переменным*(Листинг 3)*

Листинг 3. Применение метода Кохена

```
  float sum, correction_r[n][3], correction_rnt[n][3], /* correction исп. в методе
Кохена */
  correction_v[n][3], correction_vnt[n][3],
  correction_a[n][3], correction_ant[n][3];
  for(i = 0; i < n; i++)
       for(j = 0; j < 3; j++) {
            correction_r[i][j] = 0;
            correction_v[i][j] = 0;
            correction_a[i][j] = 0;
       }

  /* ГЛАВНАЯ ЧАСТЬ ПРОГРАММЫ, В КОТОРОЙ ПРИ КАЖДОЙ ИТЕРАЦИИ ИСПОЛЬЗУЮТСЯ ОСНОВНЫЕ
ФОРМУЛЫ ДЛЯ ВЫЧИСЛЕНИЯ КООРДИНАТ, СКОРОСТИ И УСКОРЕНИЯ, ПРИВЕДЕННЫЕ В УСЛОВИИ
ЗАДАЧИ */
  float dt, T, test_dt, pi = 3.14159265358;
  test_dt = 2*pi*sqrt(m[1]/k[0][1])/30;
  printf("Введите dt, которое не должно превышать следующего значения - %.6f\n",
                                                                    test_dt);
  scanf("%e", &dt); /* считывание времени dt */
  printf("dt - %.6f\n", dt);
       printf("Введенное dt превышает допустимое. Полученные ответы могут
```

```
                                                  сильно отличаться от правильных.\n");
for(p = 0; p < step; T += dt, p++) {
     printf("\n Шаг № %d\n", p);

     /* ВЫЧИСЛЕНИЕ НОВЫХ КООРДИНАТ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt */
     for(i = 0; i < n; i++) {
          printf(" Координаты тела %d:\n", i);
          for(j = 0; j < 3; j++) {
               correction_rnt[i][j] = v[i][j]*dt +  a[i][j]*dt*dt/2 -
                                                        correction_r[i][j];
               sum = r[i][j] + correction_rnt[i][j];
               correction_r[i][j] = (sum - r[i][j]) - correction_rnt[i][j];
               r[i][j] = sum;
               printf("%.15f\n", r[i][j]);
          }
          //printf("\n");
     }

     /* ВЫЧИСЛЕНИЕ НОВОЙ СКОРОСТИ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt/2 */
     for(i = 0; i < n; i++) {
          /* printf(" Скорость тела %d:\n", i); */
          for(j = 0; j < 3; j++) {
               correction_vnt[i][j] = a[i][j]*dt/2 - correction_v[i][j];
               sum = v[i][j] + correction_vnt[i][j];
               correction_v[i][j] = (sum - v[i][j]) - correction_vnt[i][j];
               v[i][j] = sum;
               /* printf("%.15f, ", v[i][j]); */
          }
          /* printf("\n"); */
     }

     /* ВЫЧИСЛЕНИЕ НОВОГО УСКОРЕНИЯ ТЕЛА i В МОМЕНТ ВРЕМЕНИ T+dt */
     for(i = 0; i < n; i++) {
          //printf(" Ускорение тела %d:\n", i);
          for(j = 0; j < 3; j++) {
               at = 0;
               float corr = 0, corr_nt;
               for(q = 0; q < n; q++) {
                    if (q == i)
                         continue;
                    power = Power(r[i][j], r[q][j], k[i][q], LenVector(r, i, q),
                                                                  len_t0[i][q]);
                    corr_nt = power/m[i] - corr;
                    sum = at + corr_nt;
                    corr = (sum - at) - corr_nt;
                    at = sum;
               }
               //printf("%.15f, ", at);
               a[i][j] = at;
          }
          //printf("\n");
     }

     /* ВЫЧИСЛЕНИЕ НОВОЙ СКОРОСТИ В МОМЕНТ ВРЕМЕНИ T+dt */
     for(i = 0; i < n; i++) {
```

```
            /* printf(" Скорость тела %d:\n", i); */
            for(j = 0; j < 3; j++) {
                correction_vnt[i][j] = a[i][j]*dt/2 - correction_v[i][j];
                sum = v[i][j] + correction_vnt[i][j];
                correction_v[i][j] = (sum - v[i][j]) - correction_vnt[i][j];
                v[i][j] = sum;
                /* printf("%.15f, ", v[i][j]); */
            }
            /* printf("\n"); */
        }
    }
```

Построим график с работы программы, которая использует метод Кохена(рис. 10):

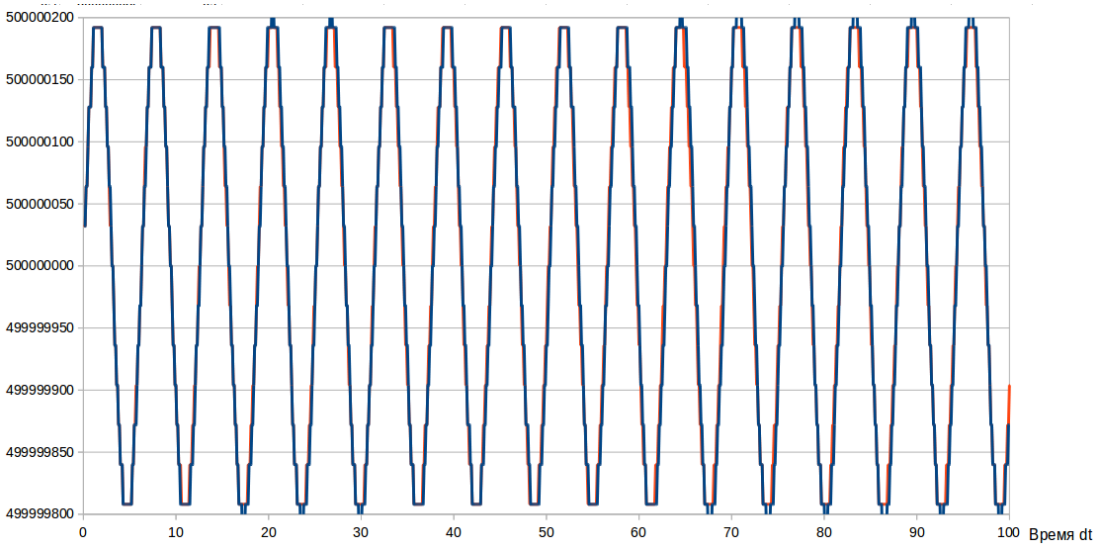

Рис. 10: Сравнение графиков колебаний при $\Delta t = 0.1$ с использованием метода Кохена.

Абсолютно точно, работа программы с использованием этого метода стала лучше, а её график(рис. 10) в точности повторяет эталонное решение. Однако для того, чтобы в дальнейшем работа программы была наиболее правильной, рекомендуется использовать числа одинакого порядка и использовать минимально возможное Δt.

Этап 4: **Исследование работы программы в случае колебания двух тел на пружинке**

Исследуем поведение нашей программы в случае колебания двух тел одинаковой массы, связанных пружинкой жесткостью $100[\frac{N}{m}]$. Для этого разместим первое тело в начало системы координат - $A\{0,0,0\}$, а второе тело отдалим от первого на радиус вектор $\bar{r}\{1,1,0\}$ - $B\{1,1,0\}$. Для того, чтобы вывести систему из положения равновесия, сообщим начальный импульс телу A - $\bar{v}_a\{1,1,0\}$. Таким образом, наши входные данные примут следующий вид:

$$
\begin{gathered}
m_1 = 1,\ m_2 = 1, \\
x_1 = 0,\ y_1 = 0,\ z_1 = 0, \\
x_2 = 1,\ y_2 = 1,\ z_2 = 0, \\
v_{1x} = 1,\ v_{1y} = 1,\ v_{1z} = 0, \\
v_{2x} = 0,\ v_{2y} = 0,\ v_{2z} = 0, \\
k = 1,\ \Delta t = 10^{-6}.
\end{gathered}
$$

С помощью них построим 3 графика - зависимость координаты y от x(рис. 11), зависимость координаты x от времени dt с использованием методом Кохена(рис. 12) и без него(рис. 13):

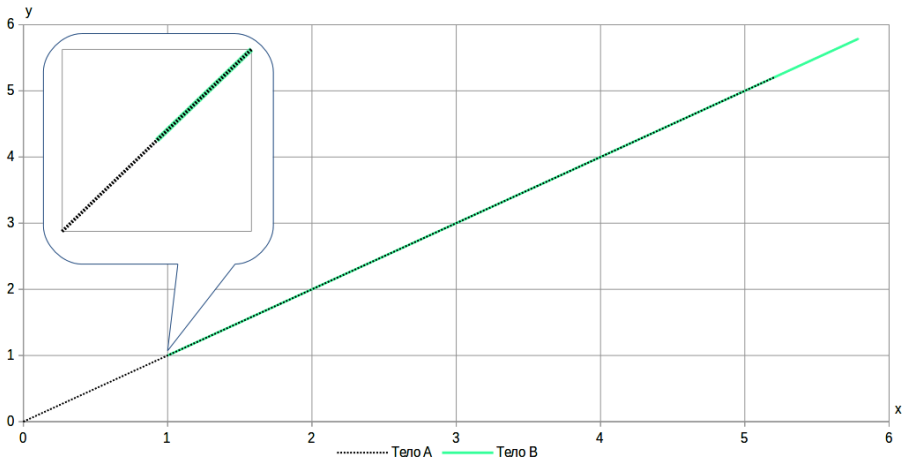

Рис. 11: Зависимость координаты y от x колебания двух тел.

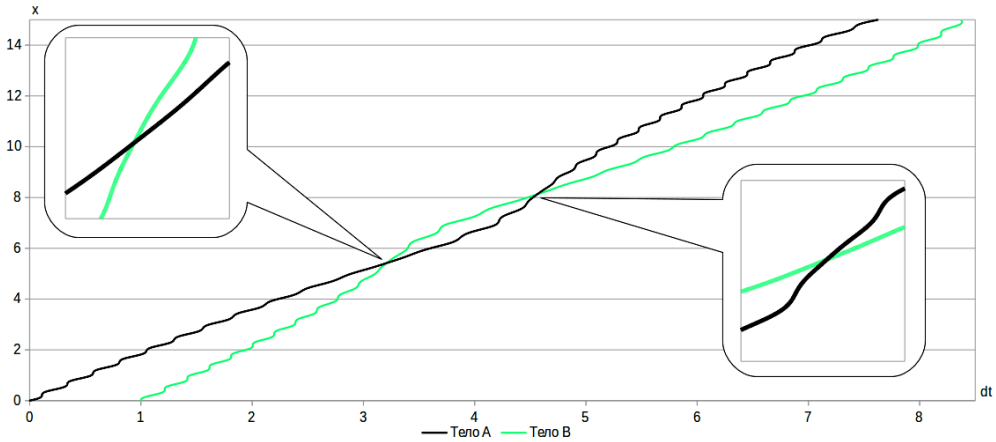

Рис. 12: Зависимость координаты x от dt колебания двух тел с использованием метода Кохена.

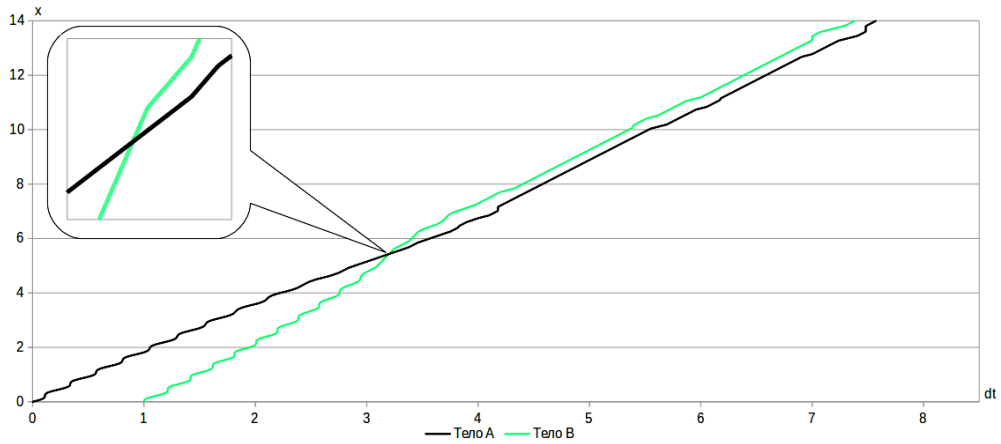

Рис. 13: Зависимость координаты x от dt колебания двух тел без использования метода Кохена.

Итак, в первом графике (рис. 11) четко видно, что колебания и движения обоих тел, как и ожидалось, происходят вдоль диагонали осей x и y, так как начальный импульс $\bar{v}_a\{1, 1, 0\}$ был сообщен телу A по направлению к телу B, а координаты обоих тел как вначале, так и на протяжении всей работы программы принадлежат графику $x = y$.

Однако интереснее взглянуть на другие два графика. Как при использовании метода Кохена(рис. 12), так и без него(рис. 13) вначале наблюдаются тихие колебания, но после третьей секунда графики пересекаются, то есть получается так, что тело A догоняет тело B, проходит "сквозь него" и встает на его место. Также удивительно то, что без использования метода Кохена(рис. 13) после того, как тела поменялись местами, колебания больше не происходят и тела движутся линейно. Объяснения этому можно тут же найти на графике с использованием метода Кохена(рис. 12), в котором программа вычисляет каждый член суммы согласно ошибке, связанной с представлением чисел с плавающей запятой в компьютере, накопившейся на предыдущих этапах. Таким образом, тело A на $4.5c$ опять меняется местами с телом B, но далее уже наблюдаются те же самые колебания, что были в самом начале работы программы, что уже больше удовлетворяет ожидаемым результатам.

Чтобы объяснить, почему тела меняются местами, построим график зависимости проекций скоростей тел на ось x от dt(рис. 14):

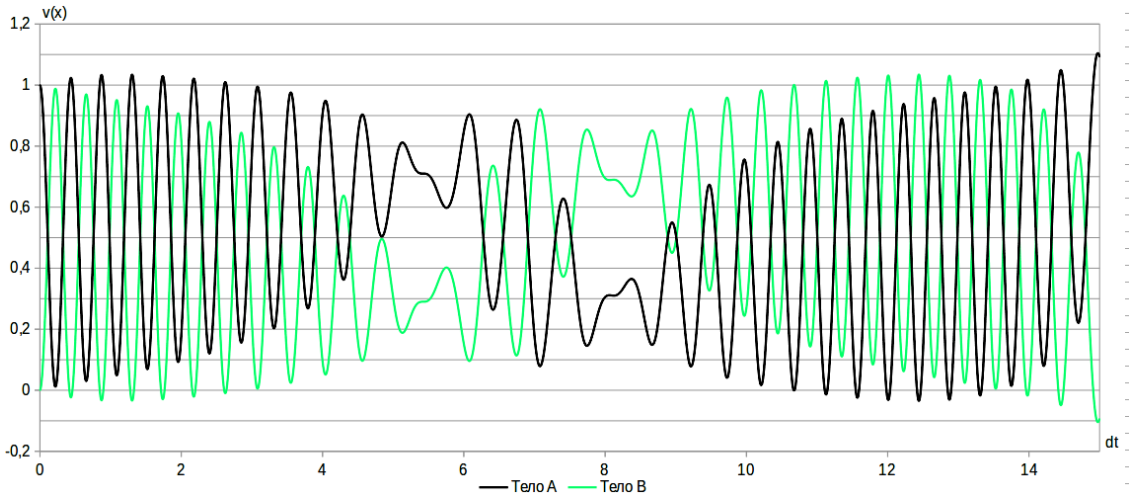

Рис. 14: Зависимость проекций соростей тел на ось x от dt.

Во-первых, стоит заметить, что проекции скоростей тел абсолютно симметричны относительно графика $y = 0.5$, что объясняет те самые колебания. Во-вторых, примерно с $4,5$ по 9 секунды график начинается очень сильно отличаться, от тех, что происходит на графиках в других промежутках вермени, то есть в те моменты, когда тела пересекается или же имеют одинаковую координату($dt = 4.5$), пружинка настолько сильно сжата, что в следующий момент придает обоим грузикам огромный импульс, но в то время, когда тела с помощью этого импульса достигнут своего амплитудного значения, пружинка начнет сжиматься и так далее, пока тела опять не будут иметь тот самый импульс, который обеспечит им легкие колебания вдоль графика $y = x$, которые наблюдались вначале и в конце графика зависимости координаты x от dt(рис. 12).