МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Университет ИТМО

ОТЧЕТ

По дисциплине:

«Программирование систем управления»

Вариант №10

ВЫПОЛНИЛ:

студент группы Е4260,

Ходжимухамедов О.И.

ПРОВЕРИЛ:

Томашевич С. И.

Санкт-Петербург

2023 г.

Оглавление

Задание....3

Исходные данные....4

1 Реализация статической модели в MATLAB....5

1.1 Определение модели вход-состояние-выход....5

1.2 Реализации модели-вход-состояние-выход....6

1.2.1 Модель в предоставлении State Space (аналоговый)....6

1.2.3 Модель в предоставлении аналоговой структурной схемы....10

1.2.4 Модель в предоставлении дискретной структурной схемы....11

2 Реализация динамической модели в MATLAB....13

2.1 Модель в представлении аналоговой структурной схемы....13

2.2 Модель в представлении дискретной структурной схемы....14

3 Реализация статической модели в C++/Qt....18

3.1 Модель вход-состояние-выход в представлении StateSpace....18

3.1 Модель вход-состояние-выход в представлении *StaticDiscreteModel*....19

4 Реализация динамической модели в C++/Qt....21

4.1 Модель в дискретном представлении....21

4.2 Модель в аналоговом представлении....23

5 Сравнения между результатами MATLAB и C++/Qt....25

Вывод....30

Приложение А....31

Приложение Б....34

Приложение В....37

Приложение Г....40

Задание

1. Преобразовать модели объекта управления в представление вход-состояние-выход
2. Дисретизировать полученную модель объекта управления с шагами дискретизации 5, 30, 100 Гц.
3. Преобразовать задающий сигнал в динамическую систему и повторить предыдущие пункты.
4. Реализовать класс интегратора в ∗.cpp и ∗.h файлах
5. Привести задающее воздействие в виде модели с использованием интегратором.
6. Программно реализовать отдельными классами четыре случая объектов (непрерывный и три дискретных). Для дискретных случаев сделать реализацию с использованием разностных уравнений (xk+1 = Axk), то есть интегратор заменяется на элемент памяти.
7. Добавить реализованные классы в предоставленную программу для QtCreator.
8. Поочередно провести сравнение поведений реализованных непрерывных моделей с дискретными моделями с соответствующими шагами дискретизации. Шаг дискретизации меняется в предоставленной программе. В результате должно получиться три пары сравнений

Исходные данные

1. Статическая модель. Передаточная функция:

$$\frac{y(t)}{u(t)} = \frac{x^3 + 4s^2 + 4s + 2}{s^3 + 4s^2 + 4s + 1}$$

2. Динамическая модель:

$$u(t) = \cos(-0.5t)$$

1 Реализация статической модели в MATLAB

1.1 Определение модели вход-состояние-выход

Для реализации модели вход-состояние-выход необходимо вначале получить матрицы **A**, **B**, **C**, **D**, которые будут описывать нашу модель.

Воспользовавшись функцией *tf2ss(num, den)* мы получим необходимые матрицы. Функции необходимо передать числовые коэффициенты числителя и знаменателя исходной передаточной функции.

$$\frac{y(t)}{u(t)} = \frac{x^3 + 4s^2 + 4s + 2}{s^3 + 4s^2 + 4s + 1}$$

Отсюда получаем следующие матрицы:

$$A = \begin{bmatrix} -4 & -4 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$C = [0 \quad 0 \quad 1]$$

$$D = [1]$$

Таким образом, наша модель вход-состояние-выход будет следующим:

$$\dot{x} = \begin{bmatrix} -4 & -4 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u$$

$$y = [0 \quad 0 \quad 1]\, x + [1]\, u$$

1.2 Реализации модели-вход-состояние-выход

1.2.1 Модель в предоставлении State Space (аналоговый)

Для реализации данной модели была использована модель *State Space* (рисунок 1), в полях которого были заданы значения полученных моделей. На его вход подавался постоянный сигнал с амплитудой 10 (далее в других подразделах будет учитываться данная амлитуда). Выходной сигнал снимался с помощью элемента *Scope*.

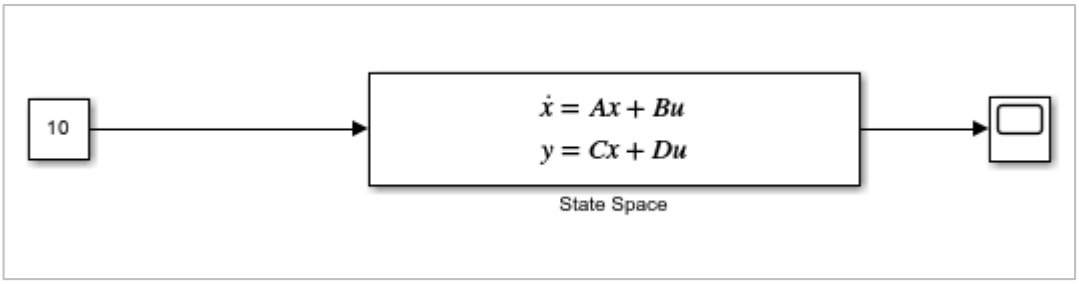

Рисунок 1 – Реализация модели вход-состояние-выход с помощью аналогового *State Space*

Выходной сигнал представлен на рисунке 2. Симуляция производилась длительность 20 секунд.

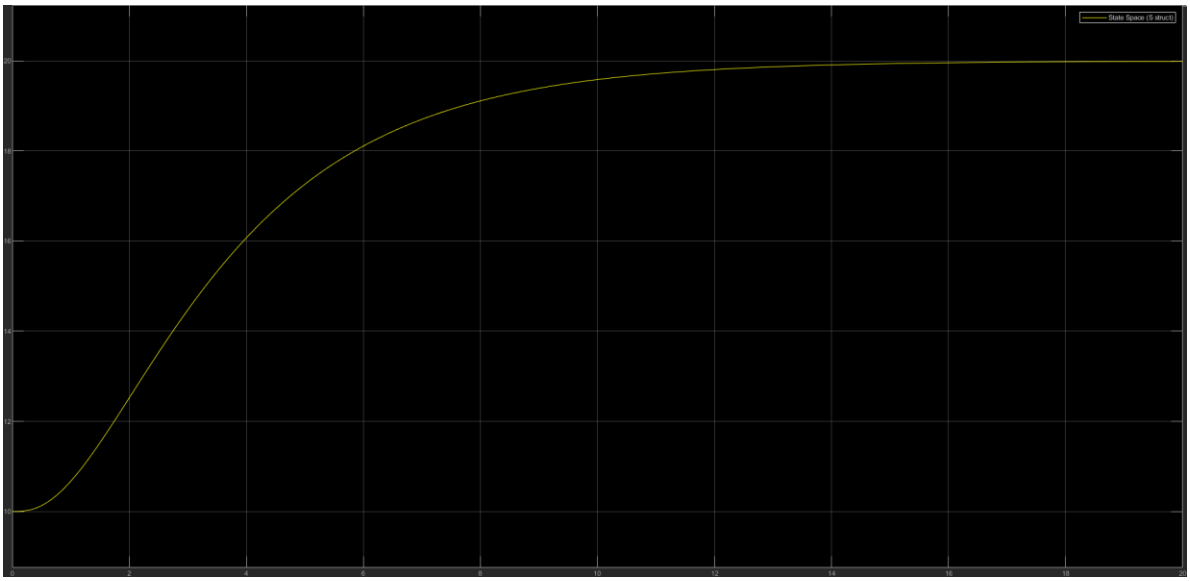

Рисунок 2 – сигнал на выходе аналогового *State Space*

1.2.2 Модель в предоставлении State Space (дискретный)

Данная модель отличается от предыдущей только блоком *State Space*, теперь она имеет дискретное представление. К тому же для данной модели требуется задать шаг дискретизации. Получив три шага дискретизаций $\Delta t_1 = 200$ мс, $\Delta t_2 = 33$ мс, $\Delta t_3 = 10$ мс, соответствующим исходным частотам, были реализованы три дискретные модели State Space. Построенная схема представлена на рисунке 3.

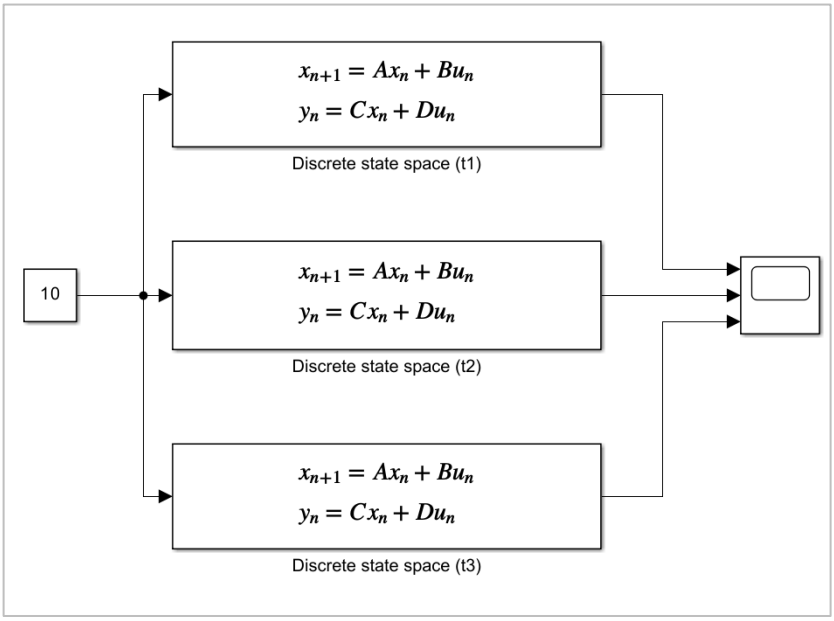

Рисунок 3 – Реализация модели вход-состояние-выход с помощью дискретного *State Space*

Для реализации модели вход-состояние-выход в дискретном виде необходимо было вначале представить матрицы системы управления в их дискретном виде. Воспользовавшись следующими уравнениями

$$\mathrm{A}_d = e^{A\Delta t}$$
$$\mathrm{B}_d = A^{-1}(\mathrm{A}_d - \mathrm{I}_n)B$$
$$\mathrm{C}_d = C$$
$$\mathrm{D}_d = D$$

$$e^{A\Delta t} = I_n + A\Delta t + \frac{(A\Delta t)^2}{2} + \cdots + \frac{(A\Delta t)^k}{k!}$$

были получены следующие матрицы:

1. Для $\Delta t_1 = 200$ мс:

$$A = \begin{bmatrix} 0.401298 & -0.55144 & 0.134015 \\ 0.1340150 & 0.937360 & -0.015385 \\ 0.0153850 & 0.195559 & 0.998904 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.134015 \\ 0.015385 \\ 0.001095 \end{bmatrix}$$

$$C = [0 \quad 0 \quad 1]$$

$$D = [1]$$

2. Для $\Delta t_2 = 33$ мс:

$$A = \begin{bmatrix} 0.873134 & -0.125265 & -0.031183 \\ 0.031183 & 0.997868 & -0.000531 \\ 0.000531 & 0.033309 & 0.999994 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.031183 \\ 0.000531 \\ 5.971131e^{-6} \end{bmatrix}$$

$$C = [0 \quad 0 \quad 1]$$

$$D = [1]$$

3. Для $\Delta t_1 = 10$ мс:

$$A = \begin{bmatrix} 0.960594 & -0.039257 & -0.009801 \\ 0.009801 & 0.999802 & -4.933830\mathrm{e}^{-5} \\ -4.933830\mathrm{e}^{-5} & 0.009999 & 0.999999 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.009801 \\ -4.933830\mathrm{e}^{-5} \\ 1.650099e^{-7} \end{bmatrix}$$

$$C = [0 \quad 0 \quad 1]$$

$$D = [1]$$

Выходные сигналы фиксировались одним Scope для наглядной разницы между моделями. Полученные графики представлены на рисунке 4.

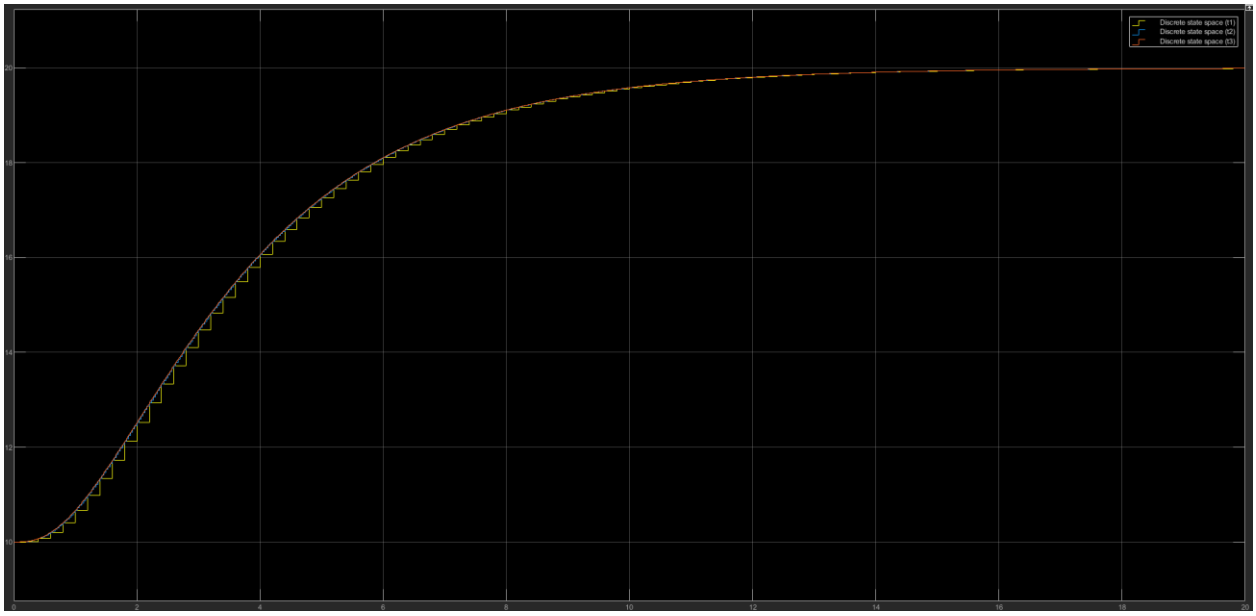

Рисунок 4 – Сигналы на выходах модели State Space

Для большей наглядности длительность симуляции была уменьшена до 5 сек. Полученные графики представлены на рисунке 5.

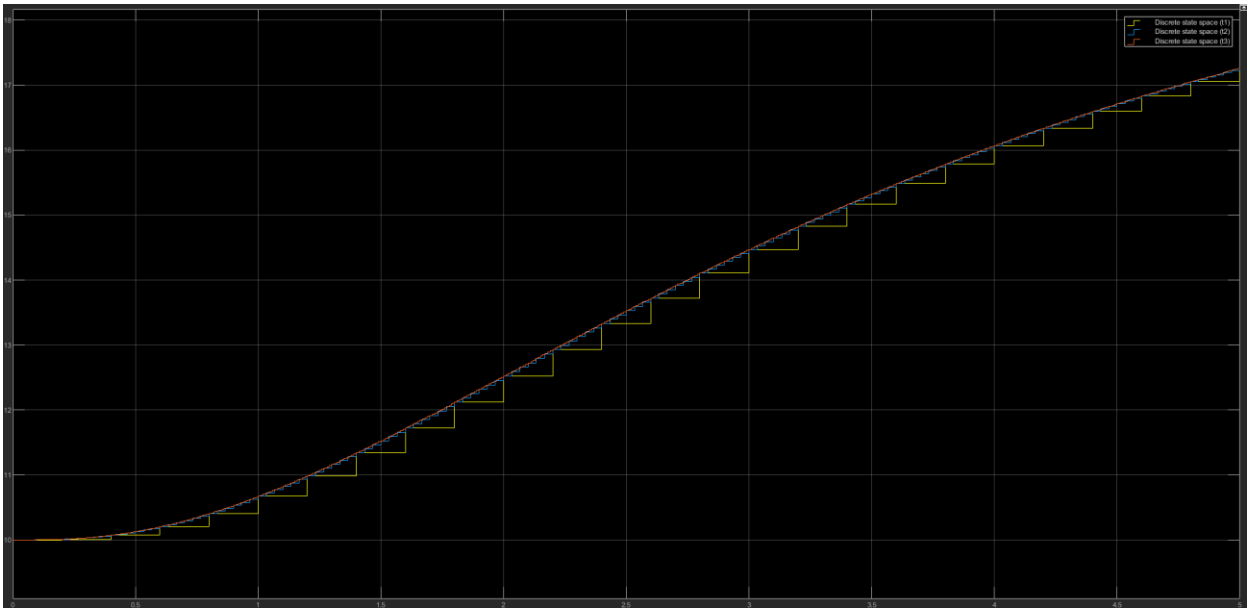

Рисунок 5 – Сигналы на выходах модели State Space при длительности симуляции 5 сек

1.2.3 Модель в предоставлении аналоговой структурной схемы

Для реализации аналоговой модели вход-состояние-выход нужно было воспользоваться такими блоками как сумматор, интегратор и усилитель. В итоге была получена схема из данных блоков, представленная на рисунке 3.

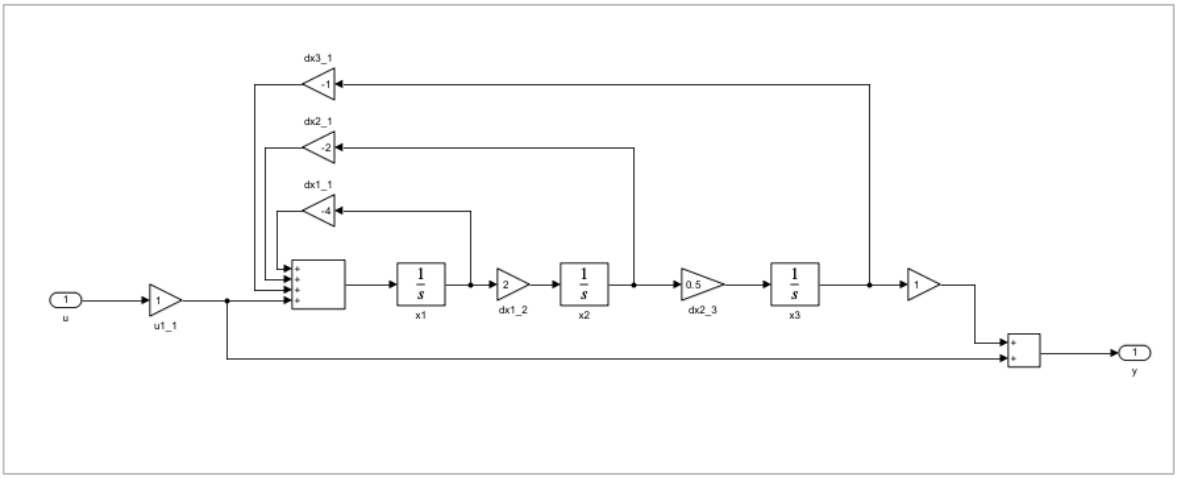

Рисунок 6 – Модель вход-состояние выход в аналоговом виде

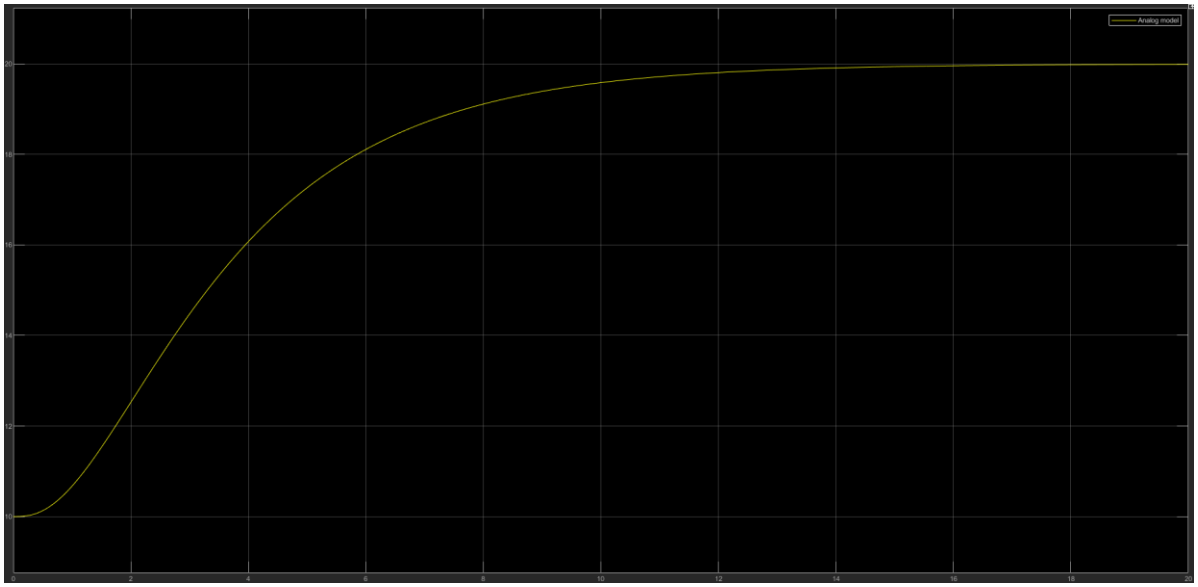

Рисунок 7 – Сигнал на выходе аналоговой модели

1.2.4 Модель в предоставлении дискретной структурной схемы

Схема дискретной модели вход-состояние-выход представлена на рисунке 8. Отличие данной схемы от аналоговой заключается в том, что теперь интеграторы заменены на блоки *Unit Delay*, задача которых задержать приходящие на них сигналы на шаг дискретизации.

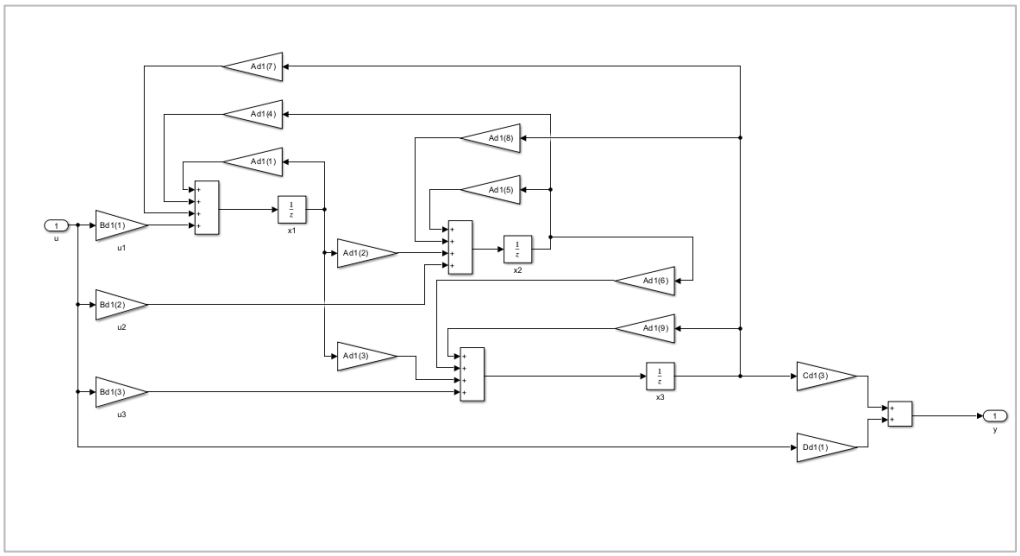

Рисунок 8 – Модель вход состояние выход в дискретном виде

Построив три таких схем для каждого шага дискретизации, получим сигналы, представленные на рисунке 9

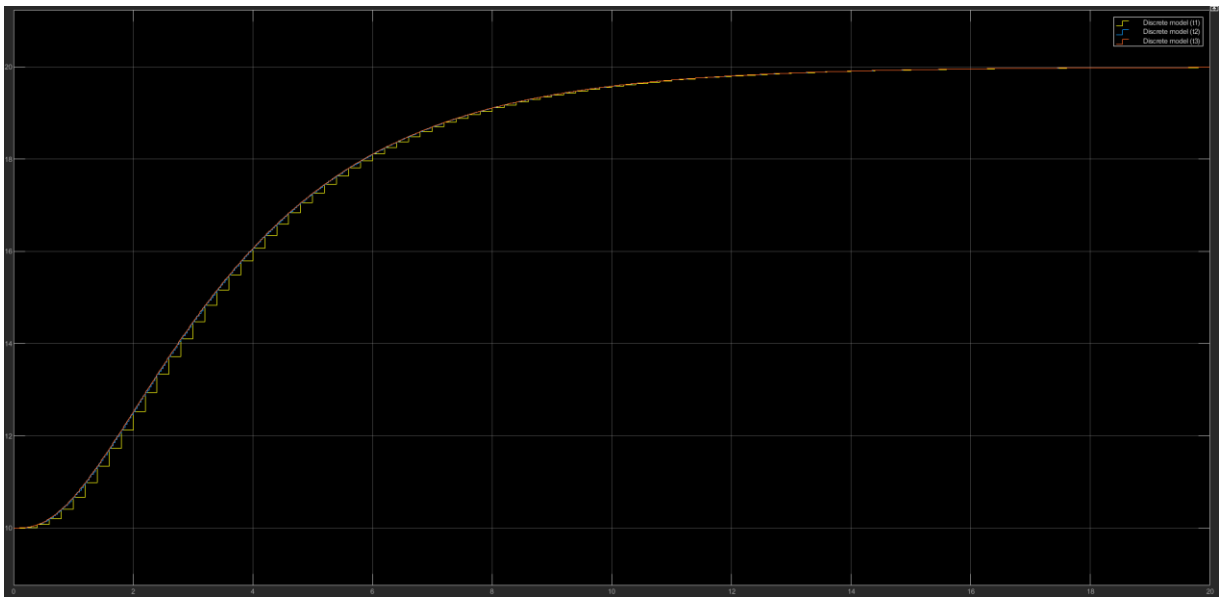

Рисунок 9 – Сигналы на выходе дискретных моделей

2 Реализация динамической модели в MATLAB

2.1 Модель в представлении аналоговой структурной схемы

Для начала необходимо было найти первую и вторую производную исходного сигнала

$$u(t) = \cos(-0.5t)$$

Выполнив операции дифференцирования, получим:

$$\dot{u}(t) = 0.5 \cdot \sin(-0.5t)$$

$$\ddot{u}(t) = -0.25 \cdot \cos(-0.5t)$$

Получив все необходимые компоненты далее были определены матрицы системы управления:

$$A = \begin{bmatrix} 0 & 1 \\ -0.25 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$C = [1 \quad 0]$$

$$D = [0]$$

Далее была построена структурная схема из интеграторов и сумматора, которая далее была сравнена с идеальным сигналом, характеристики которого соответствуют исходной модели, смещенная по оси Y на небольшое значение для наглядности.

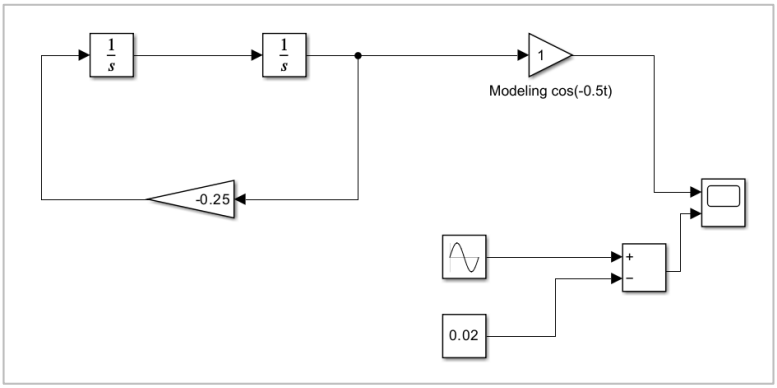

Рисунок 10 – Аналоговая структурная схема динамической модели

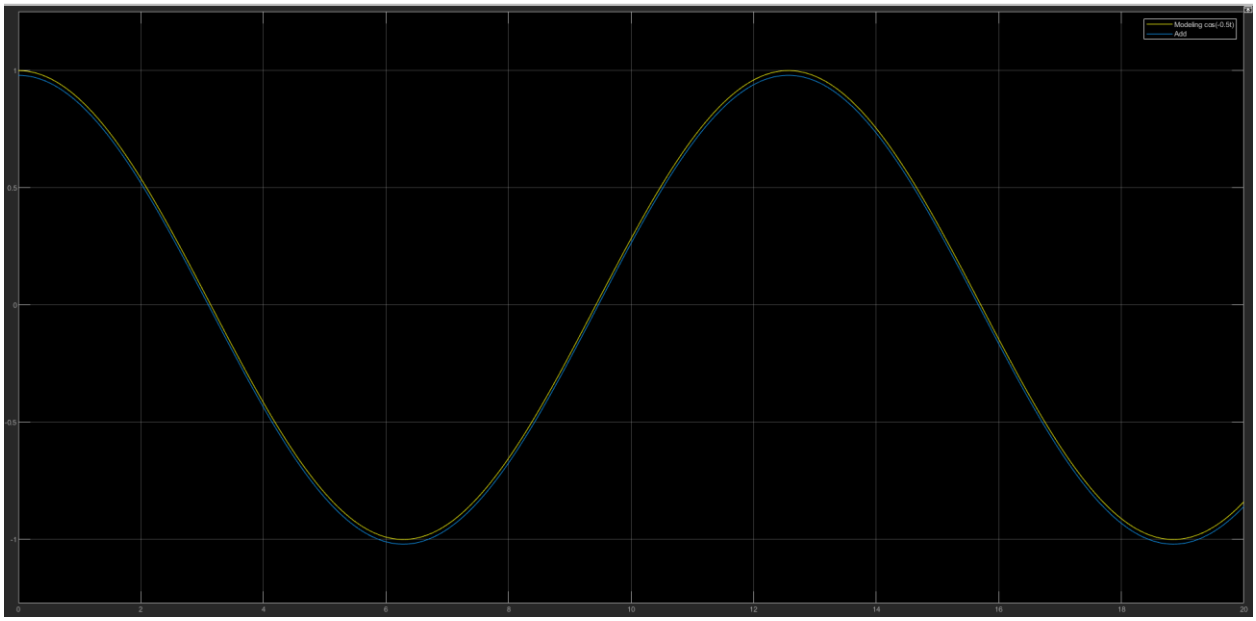

Рисунок 11 – Сравнение сигналов построенной модели и исходного сигнала

2.2 Модель в представлении дискретной структурной схемы

Для определения матриц в дискретном виде была использована функция *c2d(system_contin, dt)*, которая принимаем на вход два аргумента: моделируемую систему в аналоговом виде (которую также найдена с помощью функции ss, которой необходимо передать матрицы A, B, C, D найденные ранее) и шаг дискретизации. Таким образом, получится три группы матриц.

1. Для $\Delta t_1 = 200$ мс:

$$A = \begin{bmatrix} 0.995004 & 0.199666 \\ -0.049916 & 0.995004 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$D = [0]$$

2. Для $\Delta t_2 = 33$ мс:

$$A = \begin{bmatrix} 0.999861 & 0.033331 \\ -0.008332 & 0.999861 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$D = [0]$$

3. Для $\Delta t_1 = 10$ мс:

$$A = \begin{bmatrix} 0.999987 & 0.009999 \\ -0.002499 & 0.999987 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$D = [0]$$

Схема моделируемой системы придавлена на рисунке 12.

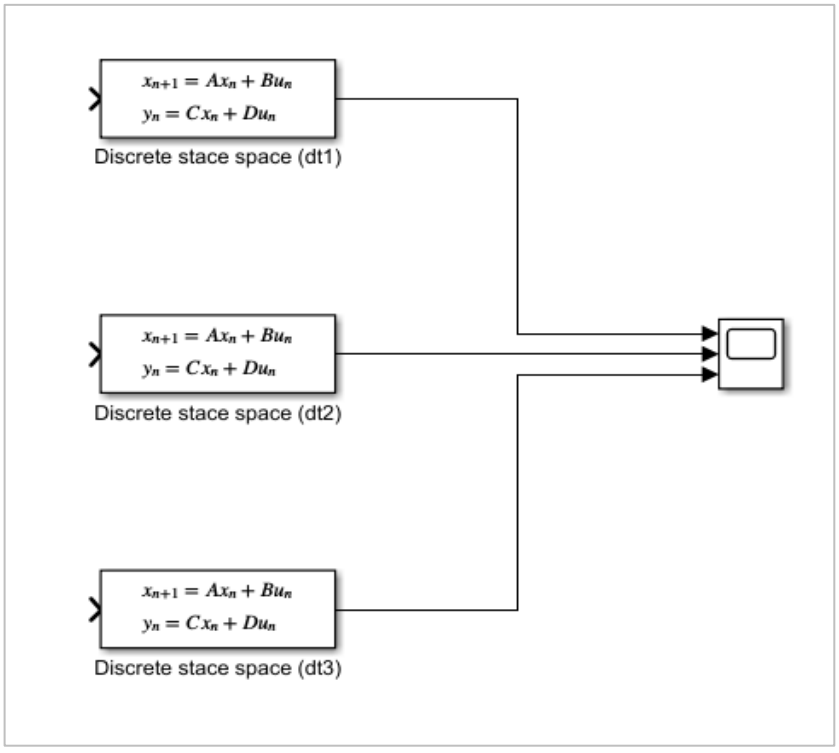

Рисунок 12 – Схема построенной модели

На рисунке 13-14 представлены графики выходных сигналов построенных моделей.

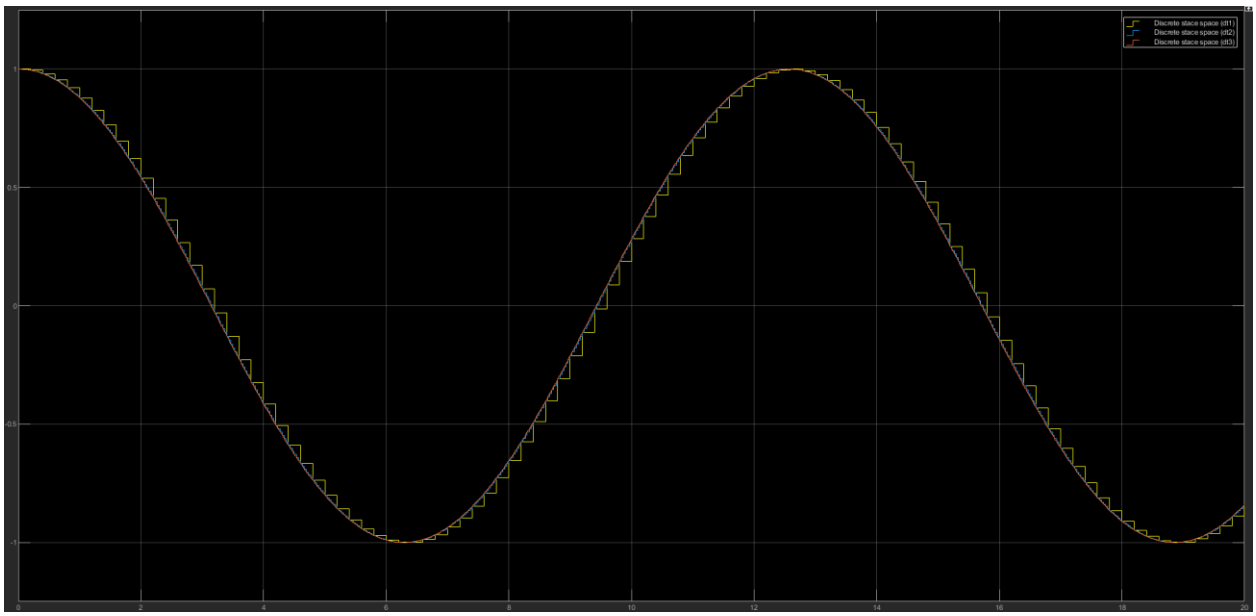

Рисунок 13 – Сигналы на выходе дискретных моделей

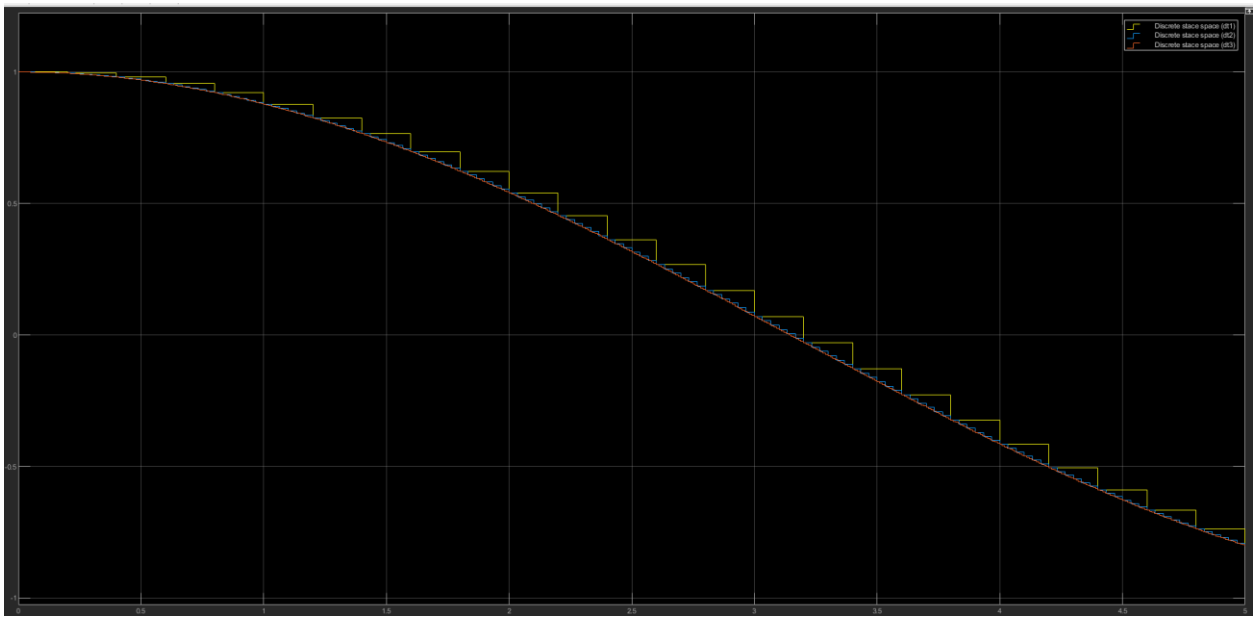

Рисунок 14 – Сигналы на выходе дискретных моделей при длительности симуляции 5 сек

3 Реализация статической модели в C++/Qt

Исходный код построенных моделей приведен в приложениях А и Б.

3.1 Модель вход-состояние-выход в представлении StateSpace

Сигналы с реализованной модели *StateSpace* представлены на рисунках 15-17.

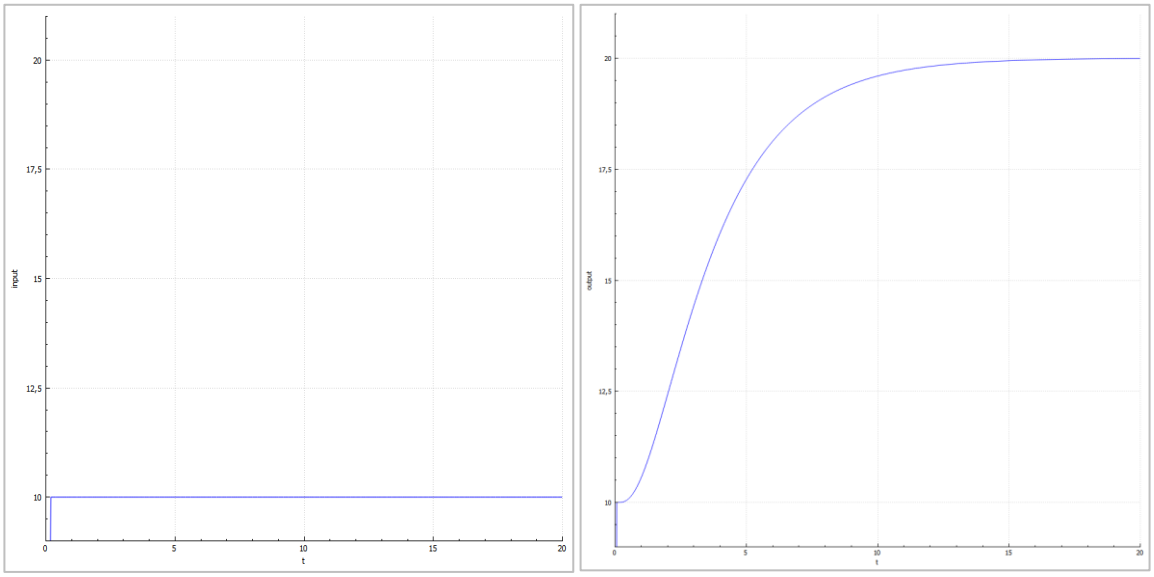

Рисунок 15 – Сигналы на входе и выходе модели *StateSpace* при частоте дискретизации 5 Гц

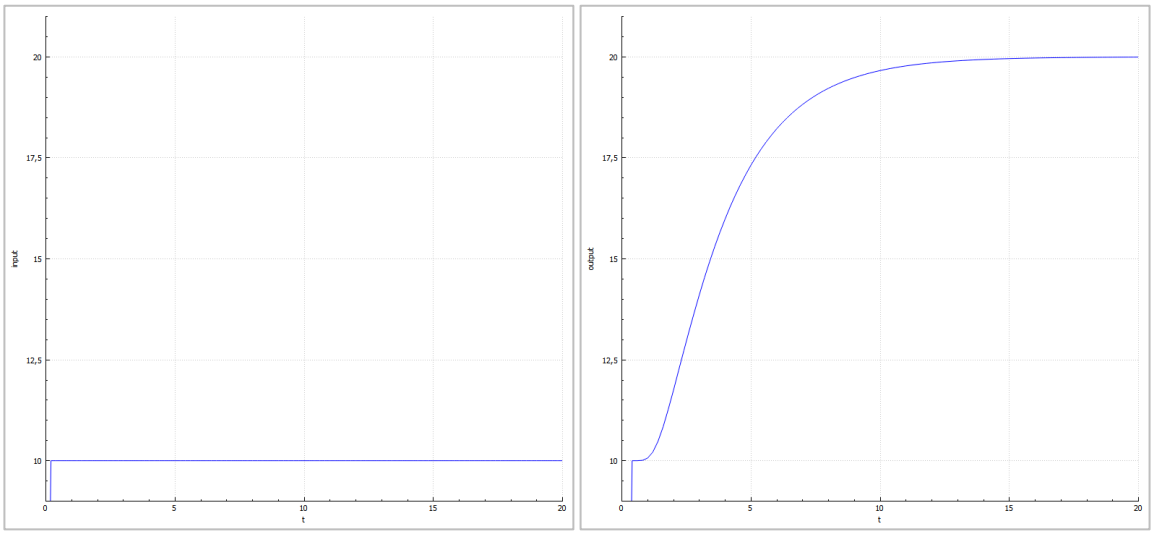

Рисунок 16 – Сигналы на входе и выходе модели *StateSpace* при частоте дискретизации 30 Гц

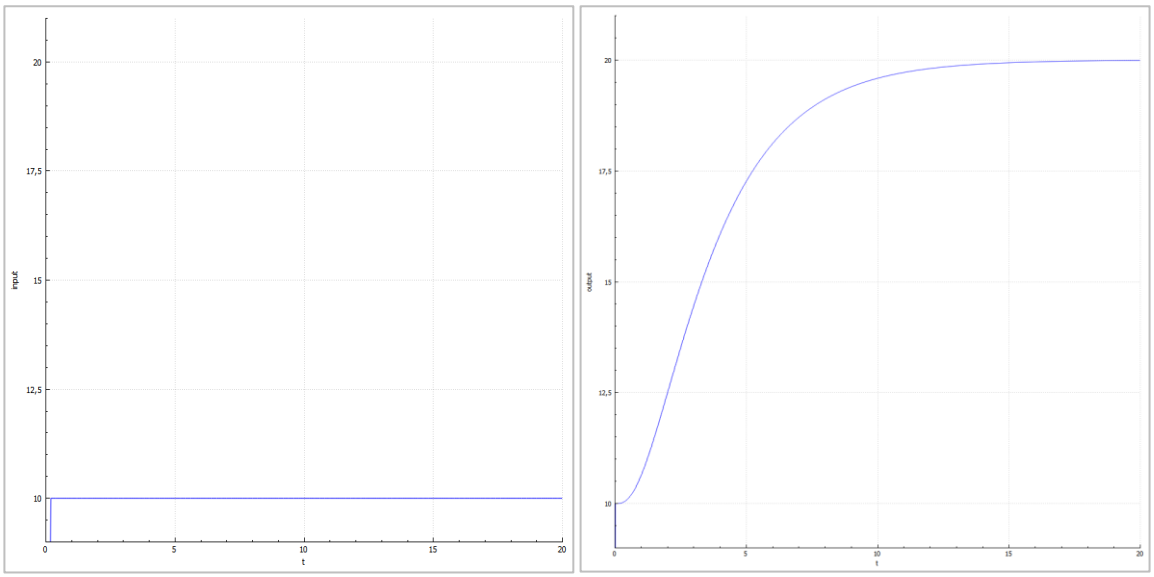

Рисунок 17 – Сигналы на входе и выходе модели *StateSpace* при частоте дискретизации 100 Гц

3.1 Модель вход-состояние-выход в представлении *StaticDiscreteModel*

Сигналы с реализованной модели *StaticDiscreteModel* представлены на рисунках 18-20.

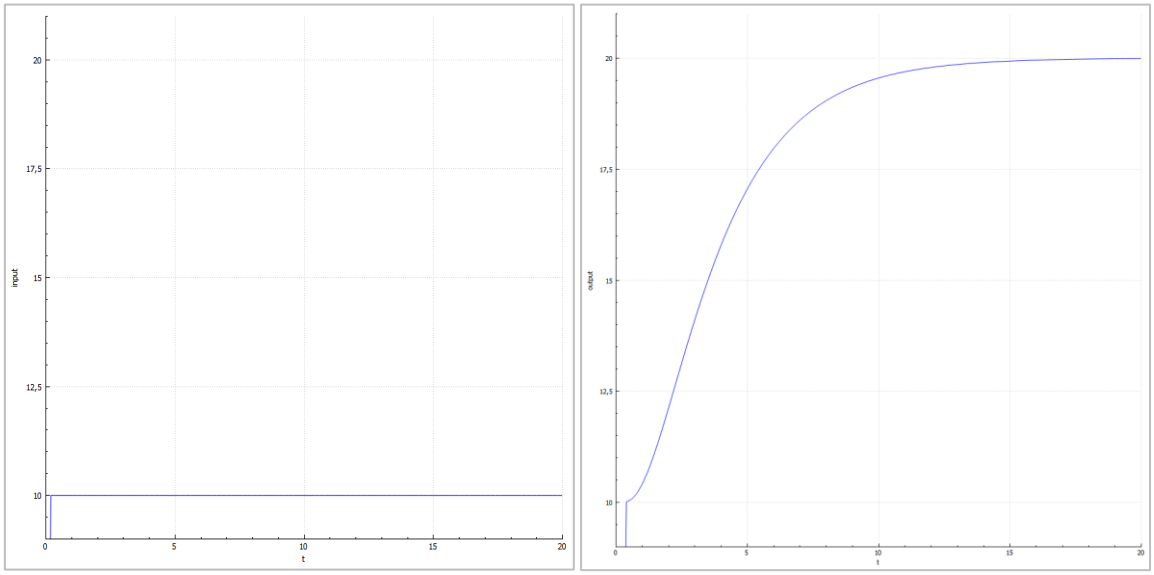

Рисунок 18 – Сигналы на входе и выходе модели *StaticDiscreteModel* при частоте дискретизации 5 Гц

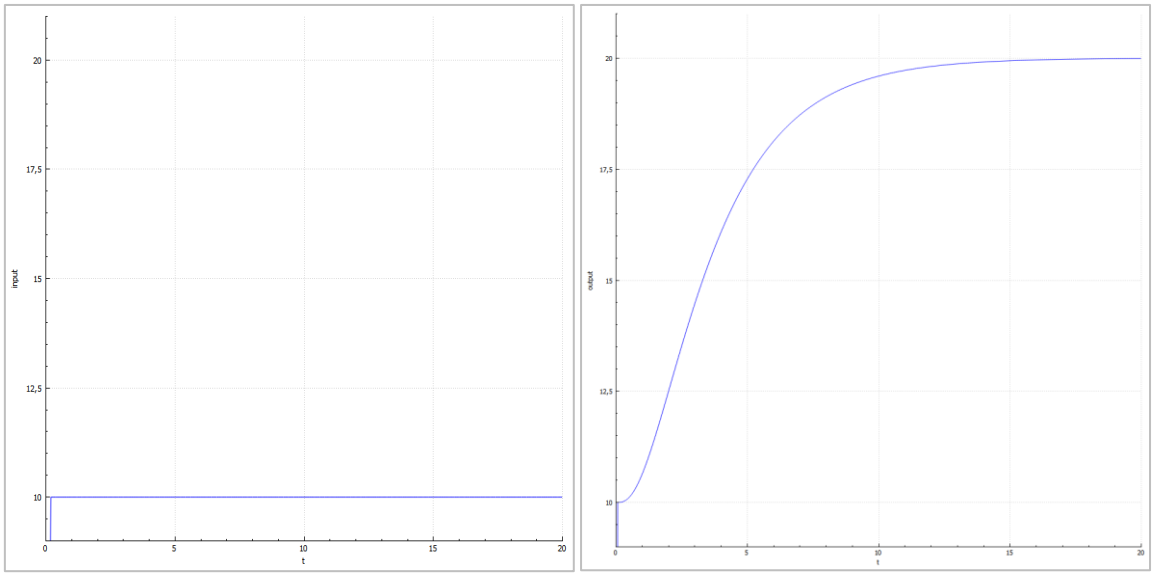

Рисунок 19 – Сигналы на входе и выходе модели *StaticDiscreteModel* при частоте дискретизации 30 Гц

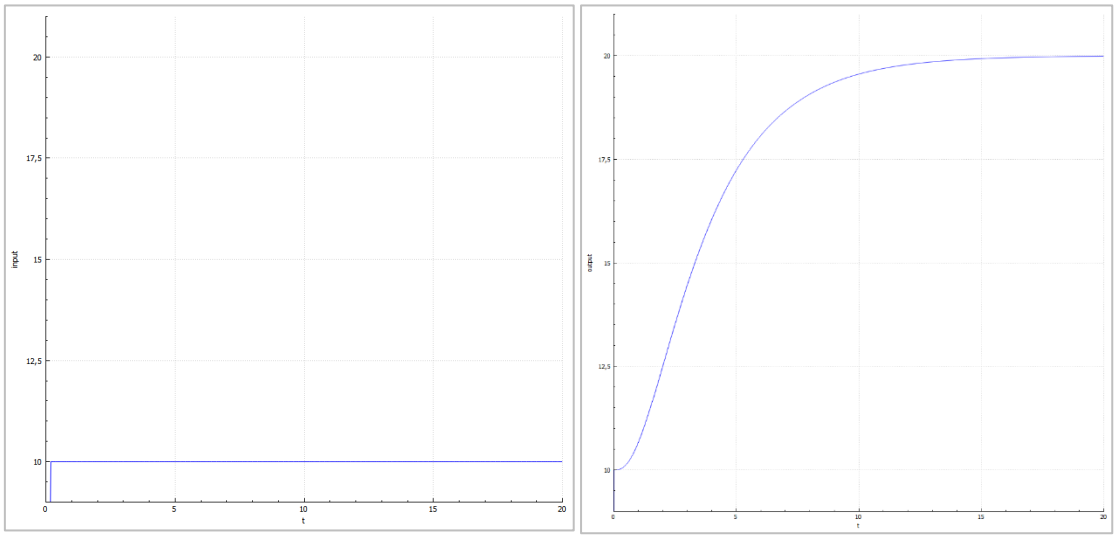

Рисунок 20 – Сигналы на входе и выходе модели *StaticDiscreteModel* при частоте дискретизации 100 Гц

4 Реализация динамической модели в C++/Qt

Исходный код построенных моделей приведен в приложениях В и Г.

4.1 Модель в дискретном представлении

Сигналы с реализованной дискретной модели *DynamicDiscreteModel* представлены на рисунках 21-23.

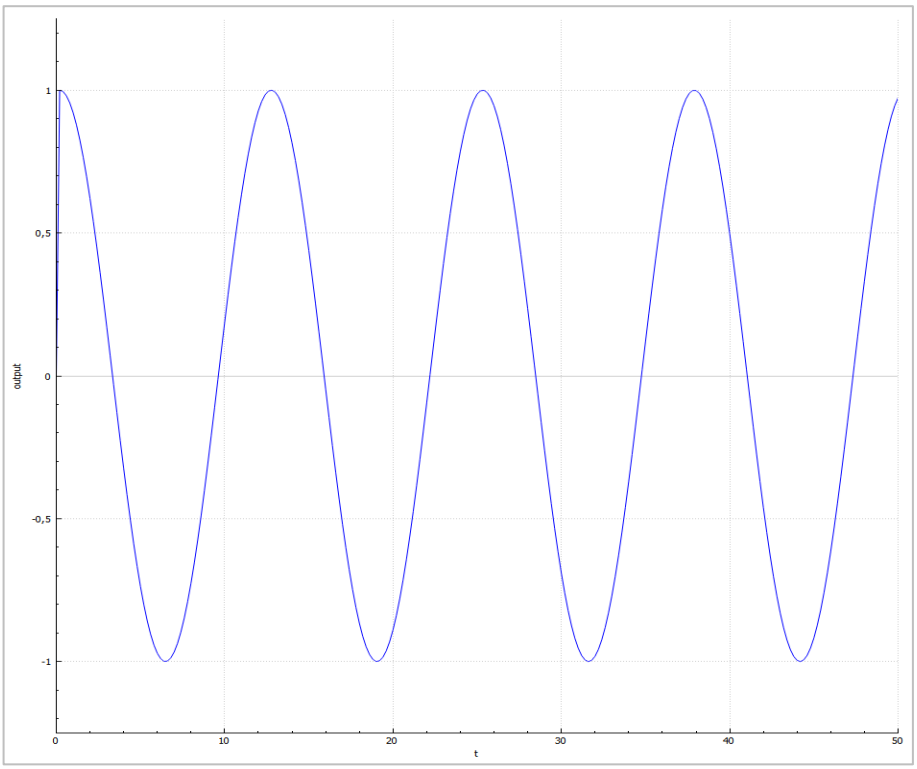

Рисунок 21 – Сигналы на выходе модели *DynamicDiscreteModel* при частоте дискретизации 5 Гц

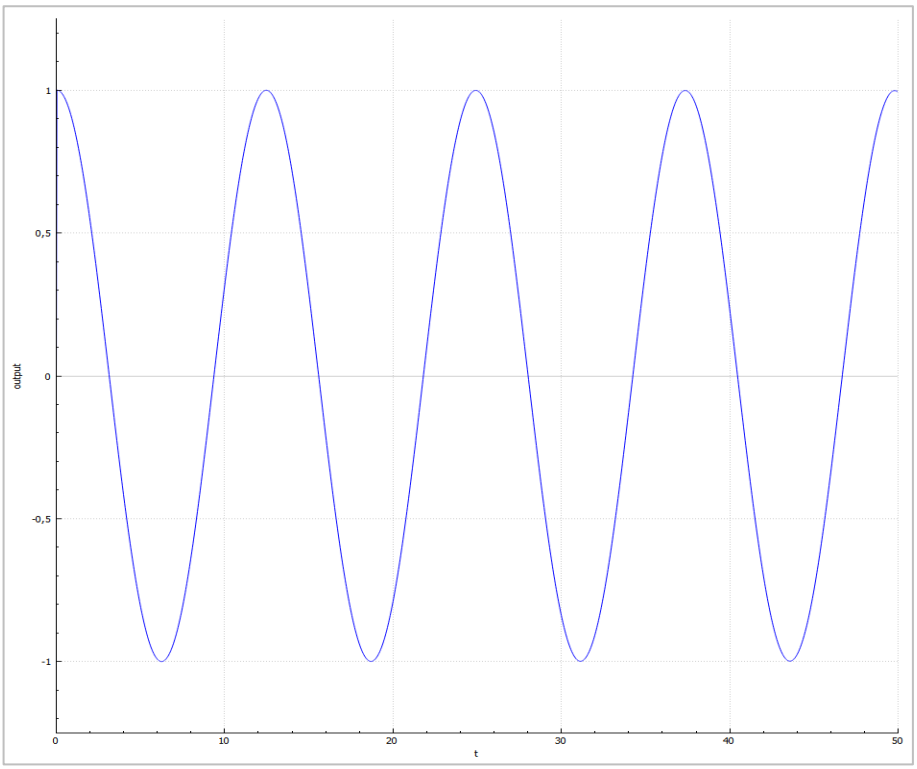

Рисунок 22 – Сигналы на выходе модели *DynamicDiscreteModel* при частоте дискретизации 33 Гц

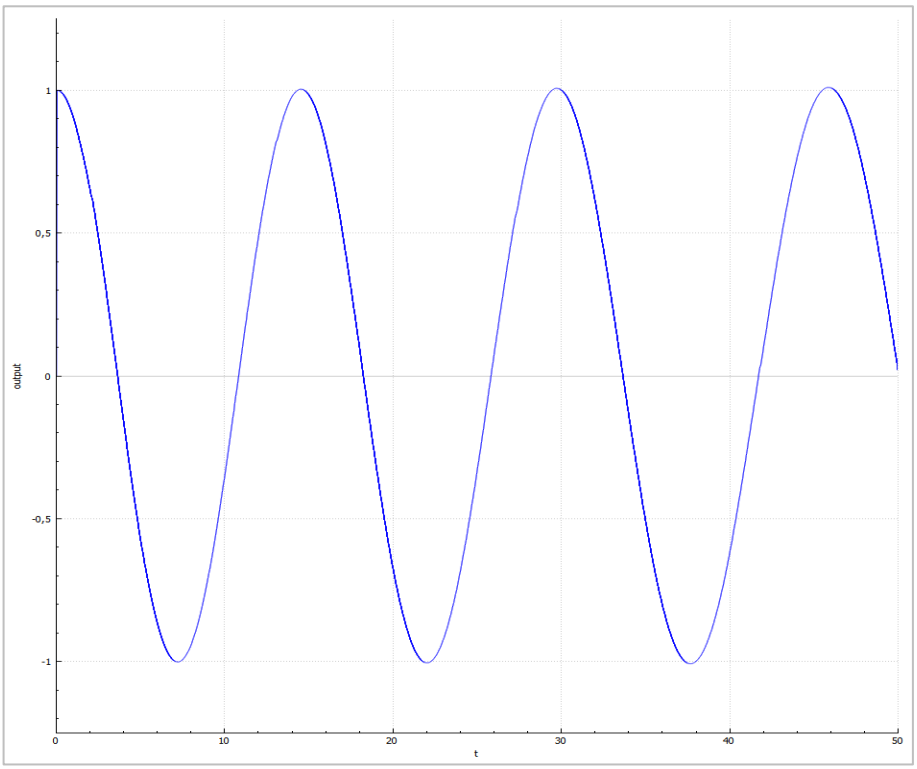

Рисунок 23 – Сигналы на выходе модели *DynamicDiscreteModel* при частоте дискретизации 100 Гц

4.2 Модель в аналоговом представлении

Сигналы с реализованной аналоговой модели *DynamicAnalogModel* представлены на рисунках 24-26.

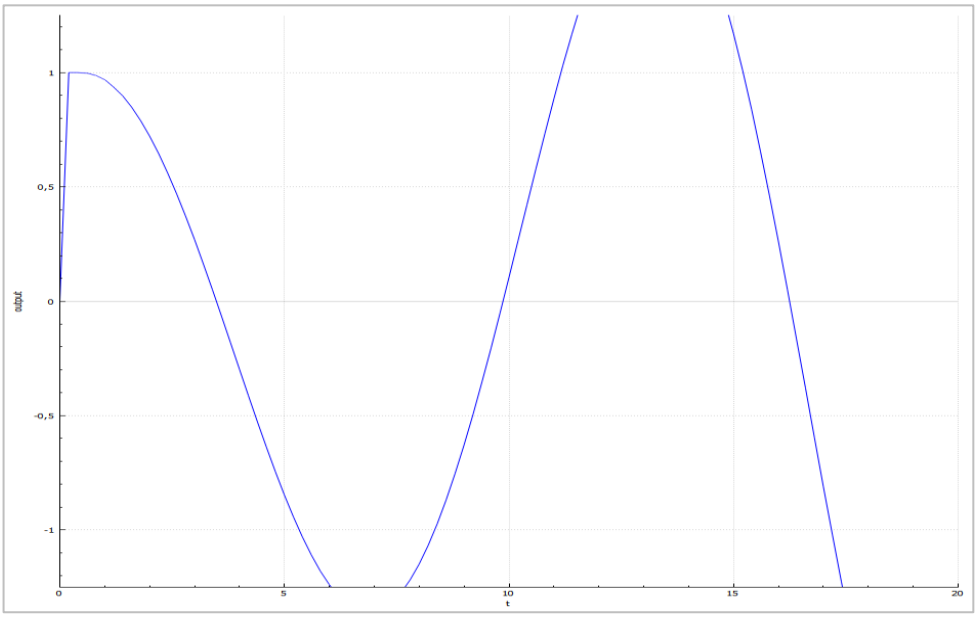

Рисунок 24 – Сигналы на выходе модели *DynamicAnalogModel* при частоте дискретизации 5 Гц

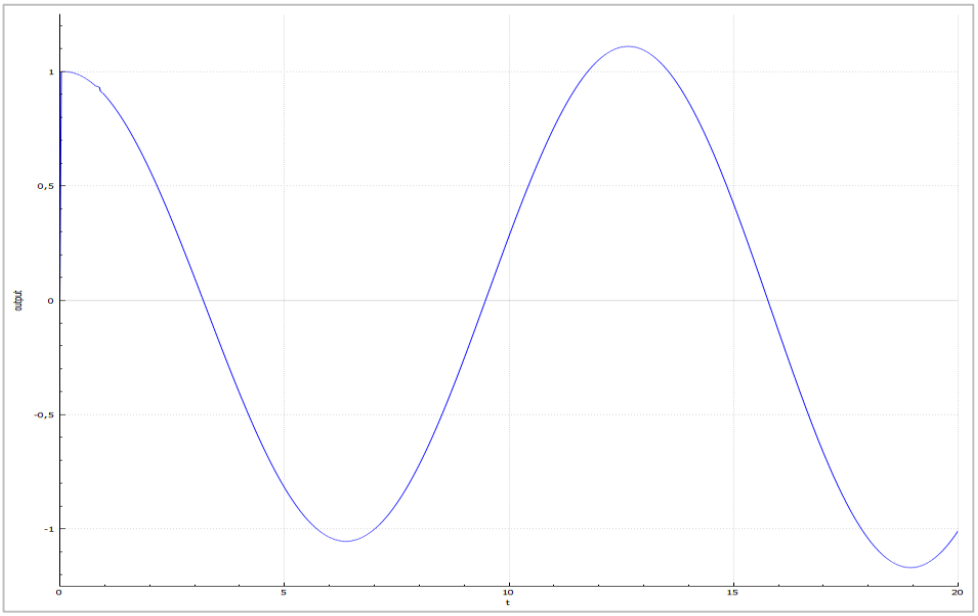

Рисунок 25 – Сигналы на выходе модели *DynamicAnalogModel* при частоте дискретизации 30 Гц

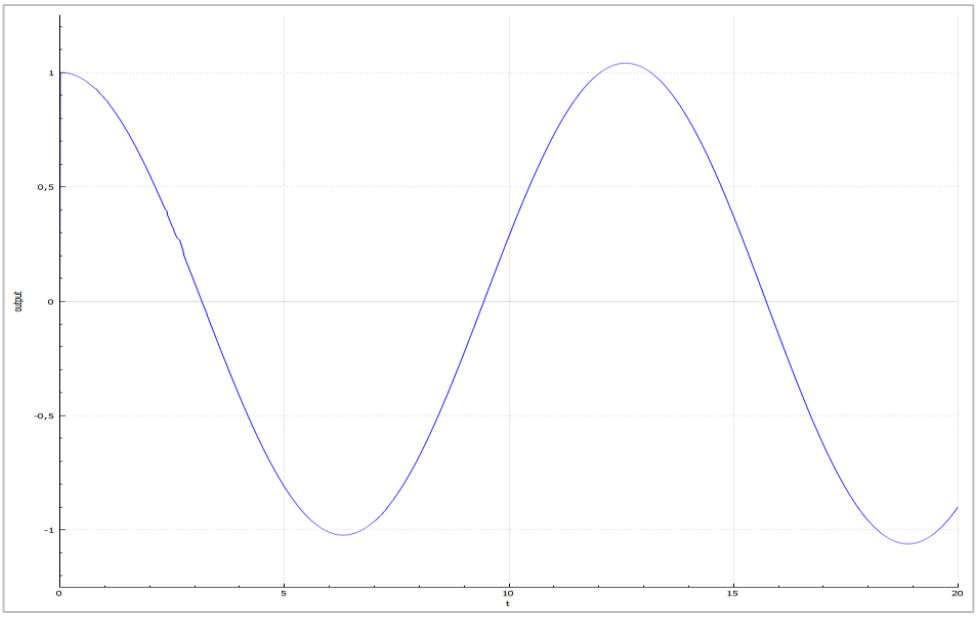

Рисунок 25 – Сигналы на выходе модели *DynamicAnalogModel* при частоте дискретизации 100 Гц

5 Сравнения между результатами MATLAB и C++/Qt

Для сравнения полученных моделей в MATLAB и C++/Qt на рисунках 26-34 приведены графики обоих реализаций.

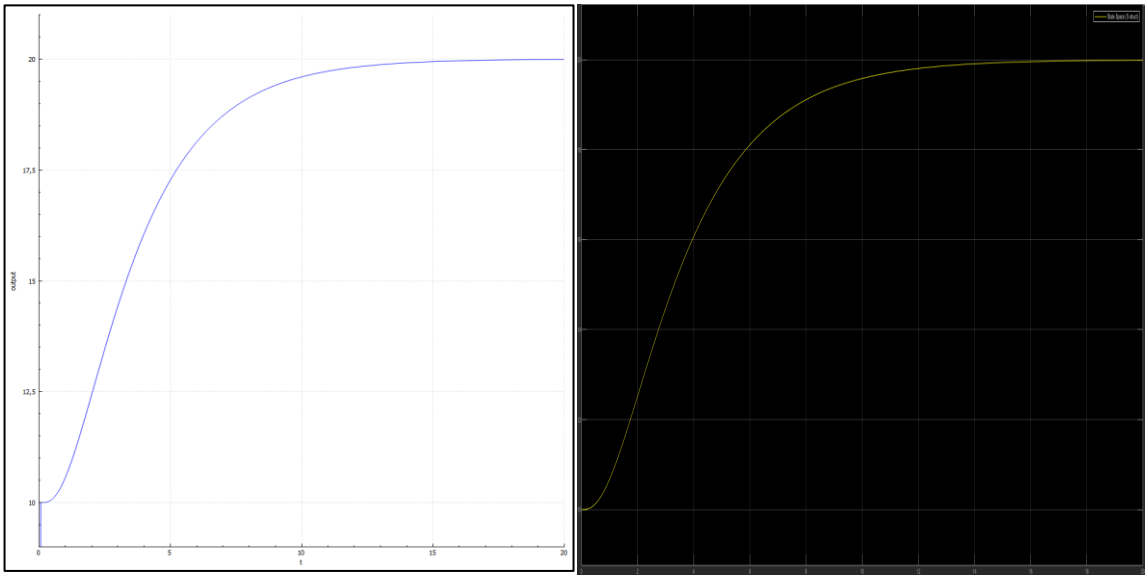

Рисунок 26 – Сигналы статических непрерывных моделей

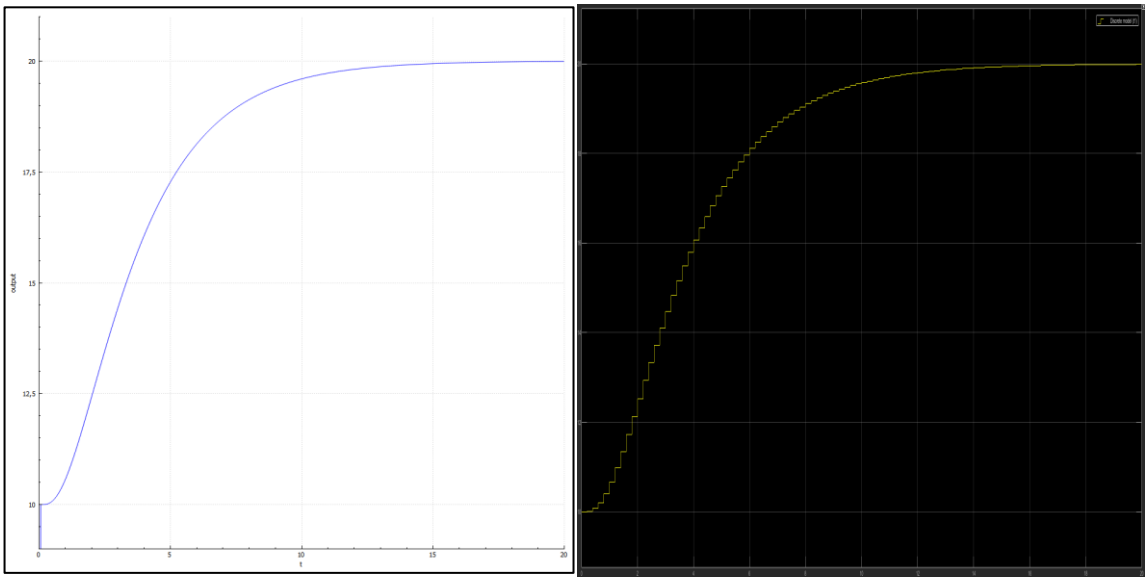

Рисунок 27 – Сигналы статических дискретных моделей при частоте 5 Гц

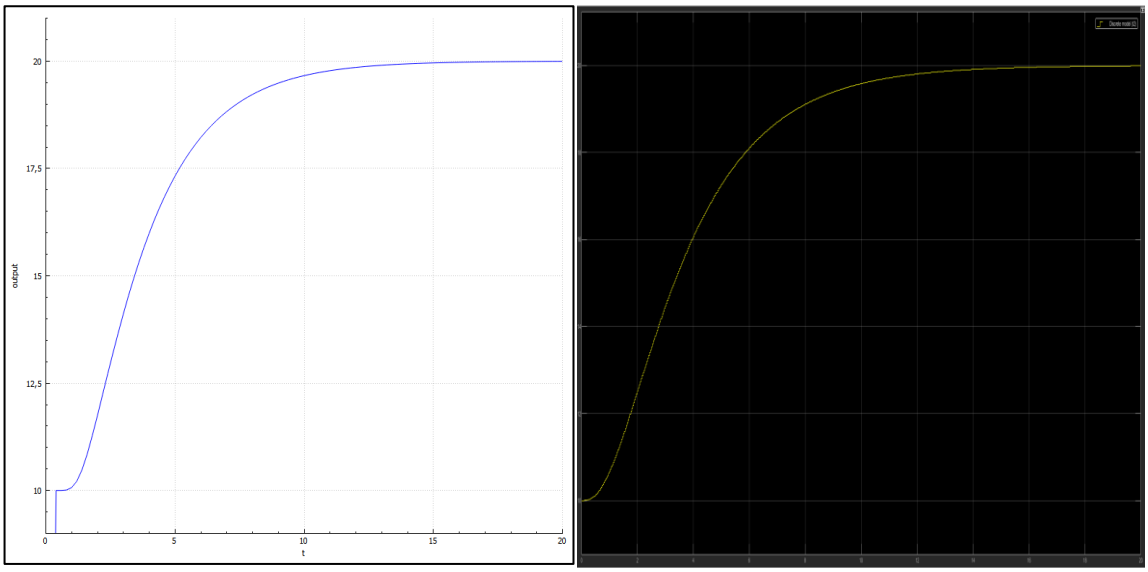

Рисунок 28 – Сигналы статических дискретных моделей при частоте 30 Гц

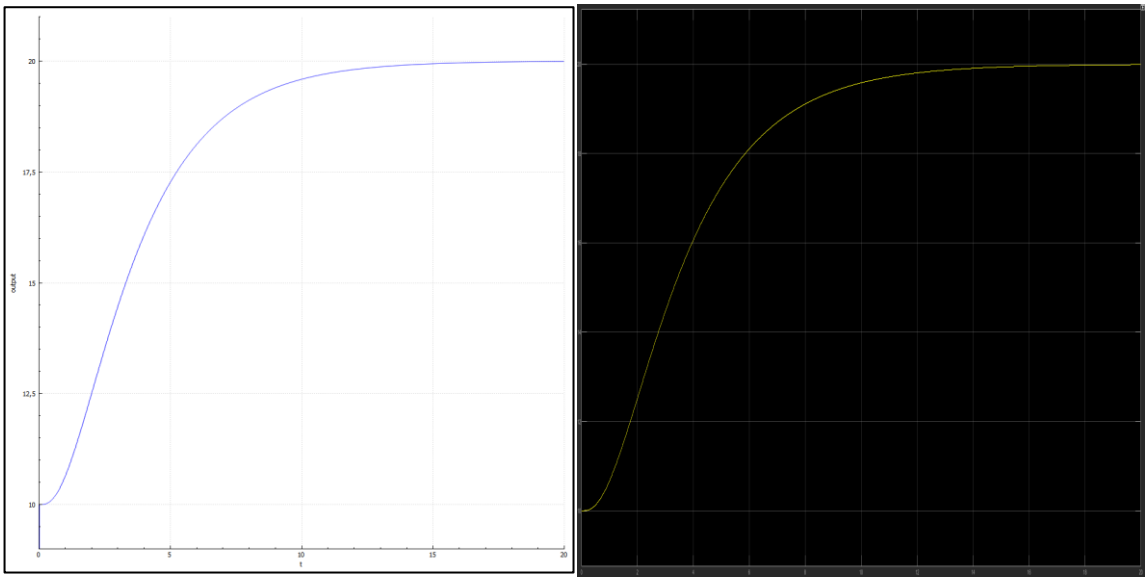

Рисунок 29 – Сигналы статических дискретных моделей при частоте 100 Гц

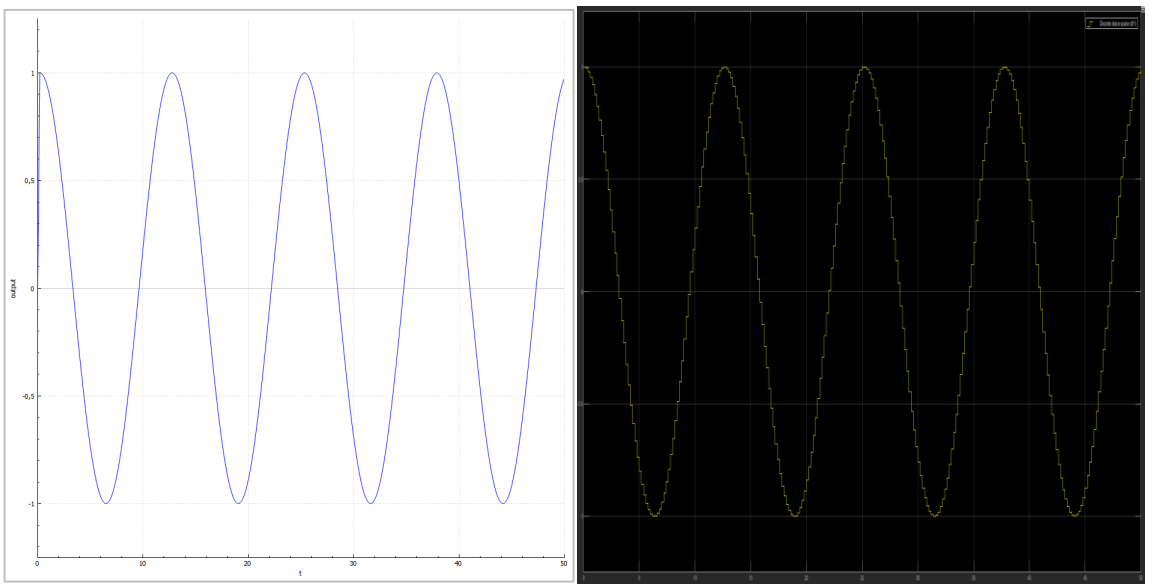

Рисунок 30 – Сигналы динамических дискретных моделей при частоте 5 Гц

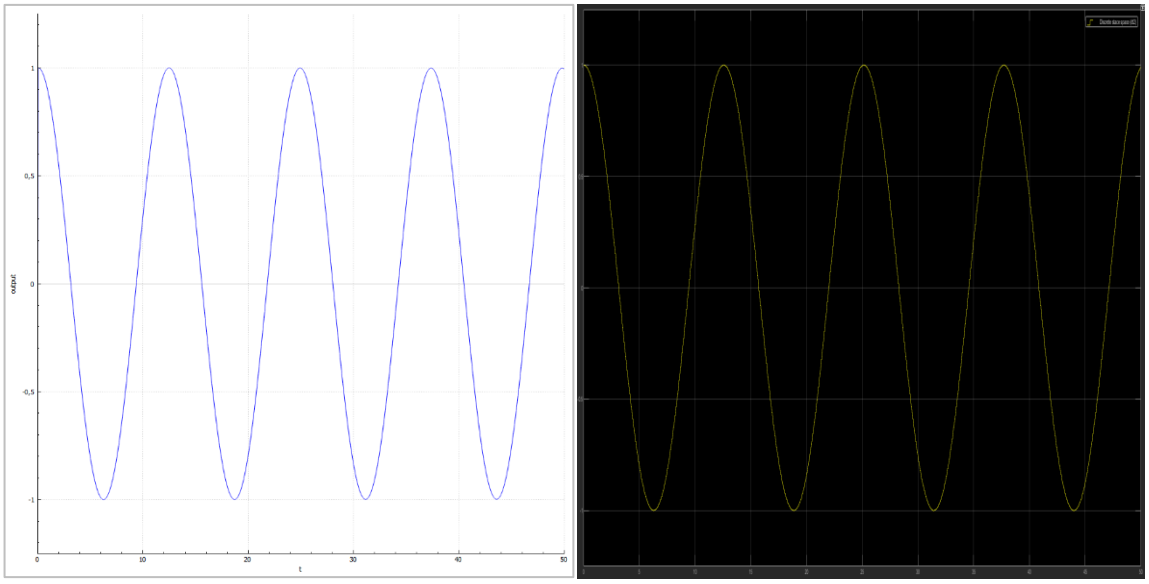

Рисунок 31 – Сигналы динамических дискретных моделей при частоте 30 Гц

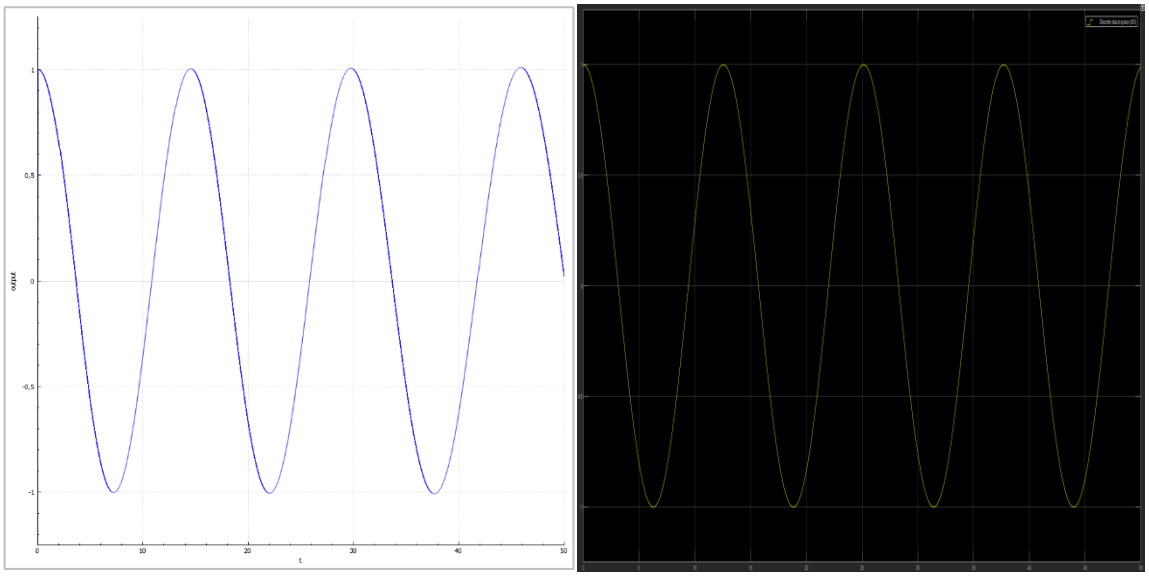

Рисунок 32 – Сигналы динамических дискретных моделей при частоте 100Гц

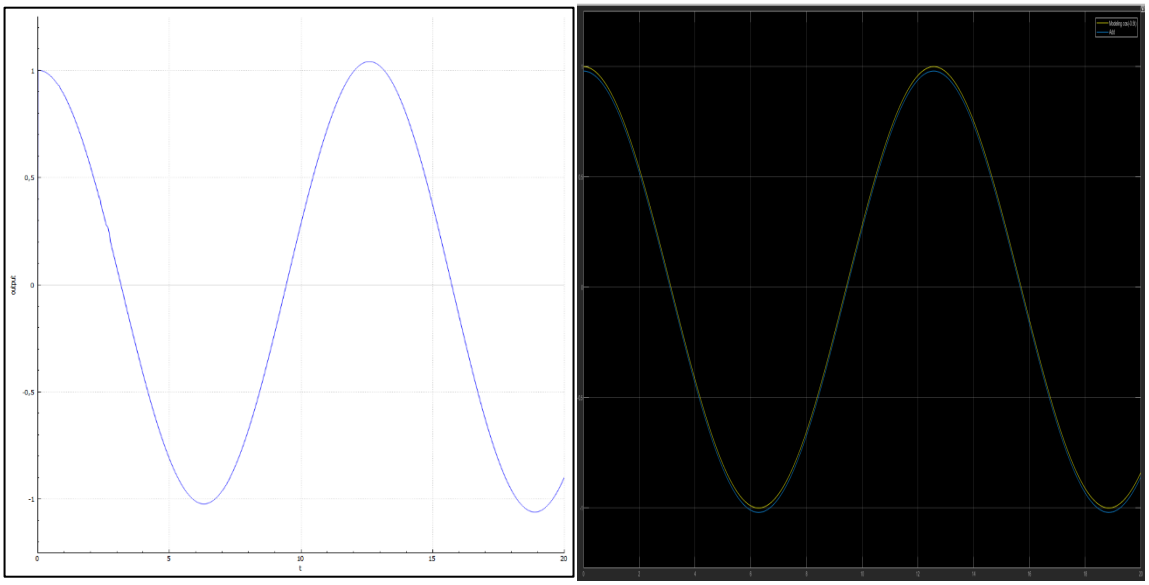

Рисунок 33 – Сигналы динамических аналоговых моделей

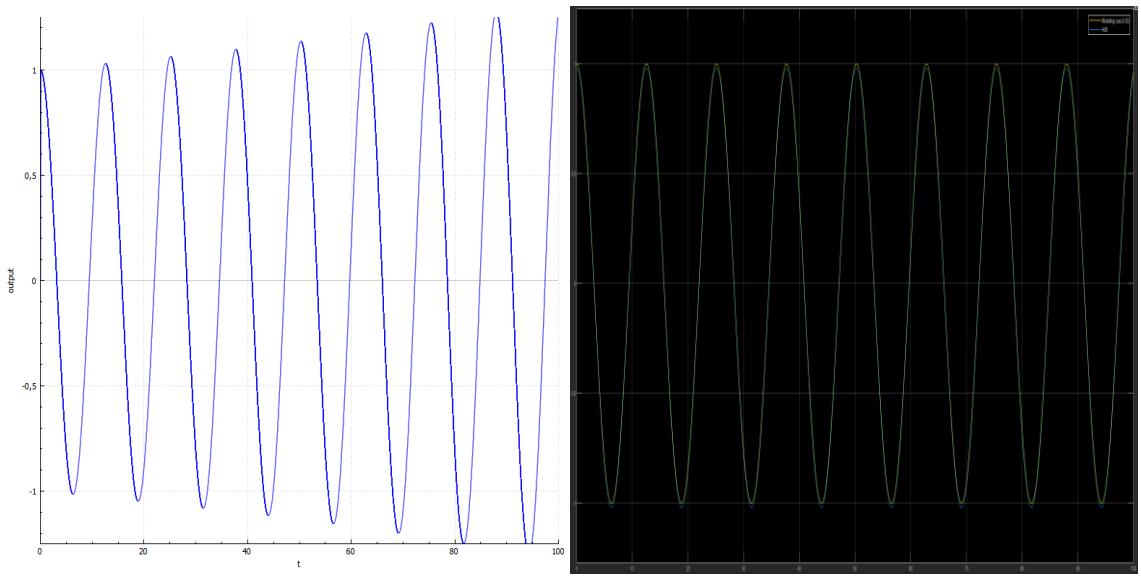

Рисунок 34 – Сигналы динамических аналоговых моделей при длительности сигнала 100 сек

Вывод

В результате выполнения практической работы были реализованы модели статической и динамической модели системы управления с помощью среды MATLAB и языка программирования C++. После этого их результаты были сверены между собой.

В ходе сравнения полученных графиков, оказалось, что реализация динамических моделей на языке программирования C++ имеет следующие недостатки. Во-первых, при реализации дискретной динамической модели, период выходного сигнала имеет зависимость от частоты дискретизации, она тем больше, чем больше частота. Во-вторых, выходной сигнал аналоговой динамической модели имеет свойство зависимости амплитуды от момента времени, она тем больше, чем больше времени прошло от начала отсчета. При увеличении частоты дискретизации скорость возрастания амплитуды остается, однако имеет малый характер, которым можно пренебречь при длительности симуляции до 15 сек.

Приложение А

StateSpace.h

```
#ifndef STATESPACE_H
#define STATESPACE_H

#include "blocks/integrator/integrator.h"
#include <vector>
#include <cstdint>

class StateSpace
{
public:
    StateSpace(std::vector<float> &m_initial_conditions,
               std::vector<std::vector<float>> &matrix_A,
               std::vector<float> &matrix_B,
               std::vector<float> &matrix_C,
               std::vector<float> &matrix_D);

    ~StateSpace();

    float getOutput();
    float update(float input, float dt);

private:
    Integrator *m_integrator_X1;
    Integrator *m_integrator_X2;
    Integrator *m_integrator_X3;

    float m_previous_input = 0.0;

    std::vector<std::vector<float>> m_matrix_A;
    std::vector<float> m_matrix_B;
    std::vector<float> m_matrix_C;
    std::vector<float> m_matrix_D;

    std::vector<float> m_initial_conditions;
};

#endif // STATESPACE_H
```

StateSpace.cpp

```
#include "statespace.h"
StateSpace::StateSpace(std::vector<float> &m_initial_conditions,
                       std::vector<std::vector<float>> &matrix_A,
                       std::vector<float> &matrix_B,
                       std::vector<float> &matrix_C,
                       std::vector<float> &matrix_D)

{
    m_integrator_X1 = new Integrator(m_initial_conditions[0]);
    m_integrator_X2 = new Integrator(m_initial_conditions[1]);
    m_integrator_X3 = new Integrator(m_initial_conditions[2]);

    m_matrix_A.resize(3);

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        m_matrix_A[i].resize(3);
    }

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        for(uint64_t j = 0; j != matrix_A[i].size(); ++j){
            m_matrix_A[i][j] = matrix_A[i][j];
        }
    }

    for(uint64_t i = 0; i != matrix_B.size(); ++i){
        m_matrix_B.push_back(matrix_B[i]);
    }

    for(uint64_t i = 0; i != matrix_C.size(); ++i){
        m_matrix_C.push_back(matrix_C[i]);
    }

    for(uint64_t i = 0; i != matrix_D.size(); ++i){
        m_matrix_D.push_back(matrix_D[i]);
    }

}

StateSpace::~StateSpace()
{
    delete m_integrator_X1;
    delete m_integrator_X2;
    delete m_integrator_X3;
}

float StateSpace::getOutput()
{
    float itgState1 = m_integrator_X2->state();
    float itgState2 = m_integrator_X2->state();
    float itgState3 = m_integrator_X3->state();
```

```
    float out1 = itgState1 * m_matrix_C[0];
    float out2 = itgState2 * m_matrix_C[1];
    float out3 = itgState3 * m_matrix_C[2];
    float in = m_matrix_D[0] * m_previous_input;
    return out1 + out2 + out3 + in;
}

float StateSpace::update(float input, float dt)
{
    float tmp_x1 = m_integrator_X1->state();
    float tmp_x2 = m_integrator_X2->state();
    float tmp_x3 = m_integrator_X3->state();
    float output = getOutput();

    m_integrator_X1->update((m_matrix_A[0][0] * tmp_x1) + (m_matrix_A[0][1]
* tmp_x2) + (m_matrix_A[0][2] * tmp_x3) + (m_matrix_B[0] * input), dt);
    m_integrator_X2->update((m_matrix_A[1][0] * tmp_x1) + (m_matrix_A[1][1]
* tmp_x2) + (m_matrix_A[1][2] * tmp_x3) + (m_matrix_B[1] * input), dt);
    m_integrator_X3->update((m_matrix_A[2][0] * tmp_x1) + (m_matrix_A[2][1]
* tmp_x2) + (m_matrix_A[1][2] * tmp_x3) + (m_matrix_B[2] * input), dt);

    m_previous_input = input;

    return output;
}
```

Приложение Б

StaticDiscreteModel.h

```
#ifndef STATICDISCRETEMODEL_H
#define STATICDISCRETEMODEL_H

#include <vector>
#include <cstdint>

class StaticDiscreteModel
{
public:
    StaticDiscreteModel(std::vector<float> &initial_conditions,
             std::vector<std::vector<float>> &matrix_A,
             std::vector<float> &matrix_B,
             std::vector<float> &matrix_C,
             std::vector<float> &matrix_D);

    float getOutput();

    float update(float input);

private:
    std::vector<std::vector<float>> m_matrix_A;
    std::vector<float> m_matrix_B;
    std::vector<float> m_matrix_C;
    std::vector<float> m_matrix_D;

    std::vector<float> m_initial_conditions;

    float m_previous_input = 0.0;
};

#endif // STATICDISCRETEMODEL_H
```

StaticDiscreteModel.cpp

```
#include "StaticDiscreteModel.h"
#include <QDebug>
StaticDiscreteModel::StaticDiscreteModel(std::vector<float>
&initial_conditions,
                  std::vector<std::vector<float>> &matrix_A,
                  std::vector<float> &matrix_B,
                  std::vector<float> &matrix_C,
                  std::vector<float> &matrix_D)
{
    m_matrix_A.resize(3);

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        m_matrix_A[i].resize(3);
    }

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        for(uint64_t j = 0; j != matrix_A[i].size(); ++j){
            m_matrix_A[i][j] = matrix_A[i][j];
        }
    }

    for(uint64_t i = 0; i != matrix_B.size(); ++i){
        m_matrix_B.push_back(matrix_B[i]);
    }

    for(uint64_t i = 0; i != matrix_C.size(); ++i){
        m_matrix_C.push_back(matrix_C[i]);
    }

    for(uint64_t i = 0; i != matrix_D.size(); ++i){
        m_matrix_D.push_back(matrix_D[i]);
    }

    for(uint64_t i = 0; i != initial_conditions.size(); ++i){
        m_initial_conditions.push_back(initial_conditions[i]);
    }
}

float StaticDiscreteModel::getOutput()
{

    return m_initial_conditions[2] * m_matrix_C[2] + m_previous_input *
m_matrix_D[0];
}

float StaticDiscreteModel::update(float input)
{
    float output = getOutput();

    float tmp_x1 = m_initial_conditions[0];
    float tmp_x2 = m_initial_conditions[1];
    float tmp_x3 = m_initial_conditions[2];
```

```
    float x1_out1 = tmp_x1 * m_matrix_A[0][0];
    float x1_out2 = tmp_x1 * m_matrix_A[1][0];
    float x1_out3 = tmp_x1 * m_matrix_A[2][0];

    float x2_out1 = tmp_x2 * m_matrix_A[0][1];
    float x2_out2 = tmp_x2 * m_matrix_A[1][1];
    float x2_out3 = tmp_x2 * m_matrix_A[2][1];

    float x3_out1 = tmp_x3 * m_matrix_A[0][2];
    float x3_out2 = tmp_x3 * m_matrix_A[1][2];
    float x3_out3 = tmp_x3 * m_matrix_A[2][2];

    float in1 = input * m_matrix_B[0];
    float in2 = input * m_matrix_B[1];
    float in3 = input * m_matrix_B[2];

    m_initial_conditions[0] = in1 + x1_out1 + x2_out1 + x3_out1;
    m_initial_conditions[1] = in2 + x1_out2 + x2_out2 + x3_out2;
    m_initial_conditions[2] = in3 + x1_out3 + x2_out3 + x3_out3;
    m_previous_input = input;

    return output;
}
```

Приложение В

DynamicDiscreteModel.h

```
#ifndef DYNAMICDISCRETEMODEL_H
#define DYNAMICDISCRETEMODEL_H

#include <vector>
#include <cstdint>

class DynamicDiscreteModel
{
public:
    DynamicDiscreteModel(std::vector<float> &initial_conditions,
                std::vector<std::vector<float>> &matrix_A,
                std::vector<float> &matrix_B,
                std::vector<float> &matrix_C,
                std::vector<float> &matrix_D);

    float getOutput();

    float update(float input);

private:
    std::vector<std::vector<float>> m_matrix_A;
    std::vector<float> m_matrix_B;
    std::vector<float> m_matrix_C;
    std::vector<float> m_matrix_D;

    std::vector<float> m_initial_conditions;

    float m_previous_input = 0.0;
};



#endif // DYNAMICDISCRETEMODEL_H
```

DynamicDiscreteModel.cpp

```
#include "DynamicDiscreteModel.h"


DynamicDiscreteModel::DynamicDiscreteModel(std::vector<float>
&initial_conditions,
                        std::vector<std::vector<float>> &matrix_A,
                        std::vector<float> &matrix_B,
                        std::vector<float> &matrix_C,
                        std::vector<float> &matrix_D)
{
    m_matrix_A.resize(2);

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        m_matrix_A[i].resize(2);
    }

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        for(uint64_t j = 0; j != matrix_A[i].size(); ++j){
            m_matrix_A[i][j] = matrix_A[i][j];
        }
    }

    for(uint64_t i = 0; i != matrix_B.size(); ++i){
        m_matrix_B.push_back(matrix_B[i]);
    }

    for(uint64_t i = 0; i != matrix_C.size(); ++i){
        m_matrix_C.push_back(matrix_C[i]);
    }

    for(uint64_t i = 0; i != matrix_D.size(); ++i){
        m_matrix_D.push_back(matrix_D[i]);
    }

    for(uint64_t i = 0; i != initial_conditions.size(); ++i){
        m_initial_conditions.push_back(initial_conditions[i]);
    }
}

float DynamicDiscreteModel::getOutput()
{
    return (m_initial_conditions[0] * m_matrix_C[0] +
            m_initial_conditions[1] * m_matrix_C[1] +
            m_previous_input * m_matrix_D[0]);
}

float DynamicDiscreteModel::update(float input)
{
    float output = getOutput();

    float tmp_x1 = m_initial_conditions[0];
    float tmp_x2 = m_initial_conditions[1];
```

```
    m_initial_conditions[0] = input * m_matrix_B[0] + tmp_x1 *
m_matrix_A[0][0] + tmp_x2 * m_matrix_A[0][1];
    m_initial_conditions[1] = input * m_matrix_B[1] + tmp_x1 *
m_matrix_A[1][0] + tmp_x2 * m_matrix_A[1][1];

    m_previous_input = input;

    return output;
}
```

Приложение Г

DynamicAnalogModel.h

```
#ifndef ANALOGSIGNAL_H
#define ANALOGSIGNAL_H
#include <vector>
#include <cstdint>
#include "blocks/integrator/integrator.h"

class DynamicAnalogModel
{
public:
    DynamicAnalogModel(std::vector<float> &initial_conditions,
              std::vector<std::vector<float>> &matrix_A,
              std::vector<float> &matrix_B,
              std::vector<float> &matrix_C,
              std::vector<float> &matrix_D);

    ~DynamicAnalogModel();

    float getOutput();
    float update(float input, float dt);

private:
    Integrator *m_integrator_X1;
    Integrator *m_integrator_X2;

    float m_previous_input = 0.0;

    std::vector<std::vector<float>> m_matrix_A;
    std::vector<float> m_matrix_B;
    std::vector<float> m_matrix_C;
    std::vector<float> m_matrix_D;

    std::vector<float> m_initial_conditions;
};

#endif // ANALOGSIGNAL_H
```

DynamicAnalogModel.cpp

```
#include "DynamicAnalogModel.h"


DynamicAnalogModel::DynamicAnalogModel(std::vector<float>
&initial_conditions,
                    std::vector<std::vector<float>> &matrix_A,
                    std::vector<float> &matrix_B,
                    std::vector<float> &matrix_C,
                    std::vector<float> &matrix_D)

{
    m_integrator_X1 = new Integrator(initial_conditions[0]);
    m_integrator_X2 = new Integrator(initial_conditions[1]);

    m_matrix_A.resize(2);

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        m_matrix_A[i].resize(2);
    }

    for(uint64_t i = 0; i != matrix_A.size(); ++i){
        for(uint64_t j = 0; j != matrix_A[i].size(); ++j){
            m_matrix_A[i][j] = matrix_A[i][j];
        }
    }

    for(uint64_t i = 0; i != matrix_B.size(); ++i){
        m_matrix_B.push_back(matrix_B[i]);
    }

    for(uint64_t i = 0; i != matrix_C.size(); ++i){
        m_matrix_C.push_back(matrix_C[i]);
    }

    for(uint64_t i = 0; i != matrix_D.size(); ++i){
        m_matrix_D.push_back(matrix_D[i]);
    }

}

DynamicAnalogModel::~DynamicAnalogModel()
{
    delete m_integrator_X1;
    delete m_integrator_X2;
}

float DynamicAnalogModel::getOutput()
{
    return m_matrix_C[0] * m_integrator_X1->state() +
           m_matrix_C[1] * m_integrator_X2->state() +
           m_matrix_D[0] * m_previous_input;
}
```

```
float DynamicAnalogModel::update(float input, float dt)
{
    float tmp_x1 = m_integrator_X1->state();
    float tmp_x2 = m_integrator_X2->state();
    float output = getOutput();

    m_integrator_X1->update((m_matrix_A[0][0] * tmp_x1) + (m_matrix_A[0][1]
* tmp_x2)  + (m_matrix_B[0] * input), dt);
    m_integrator_X2->update((m_matrix_A[1][0] * tmp_x1) + (m_matrix_A[1][1]
* tmp_x2)  + (m_matrix_B[1] * input), dt);

    m_previous_input = input;

    return output;
}
```