**МИНОБРНАУКИ РОССИИ**

**Национальный исследовательский университет ИТМО**

отчет

**по практической работе**

**по дисциплине «Программирование систем управления»**

|  |  |  |
| --- | --- | --- |
| Студент гр. R4140 |  | Федоров И.А. |
| Преподаватель |  | Томашевич С.И. |

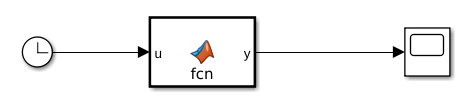
Санкт-Петербург

2022

**Вариант 16**:



**1)** Простейшая модель генератора в среде Simulink может быть реализована с помощью блока MATLAB Function:



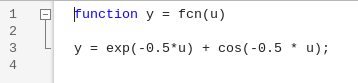


Рисунок 1 – Простейшая модель генератора

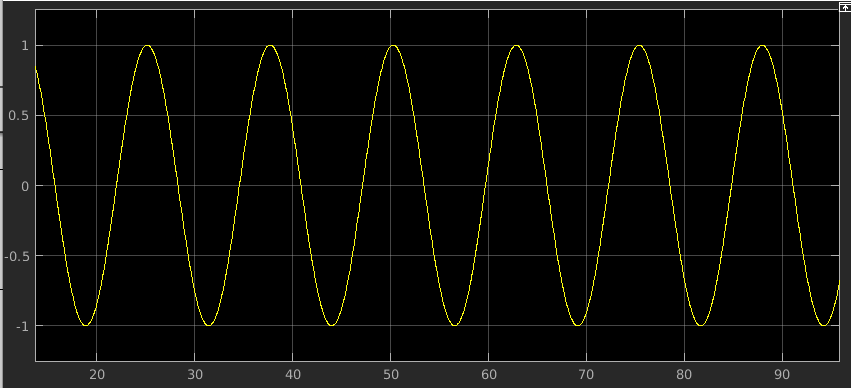


Рисунок 2 – График для генератора

**2)** Составим структурную схему для генератора. Разделим генератор на две части:

|  |  |
| --- | --- |
|  |  |

Получим следующую структурную схему:

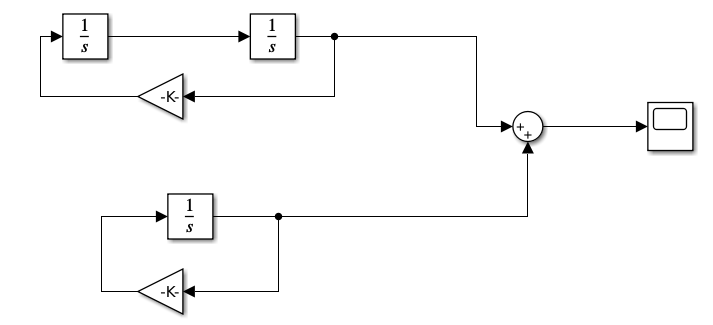


Рисунок 3 – Структурная схема

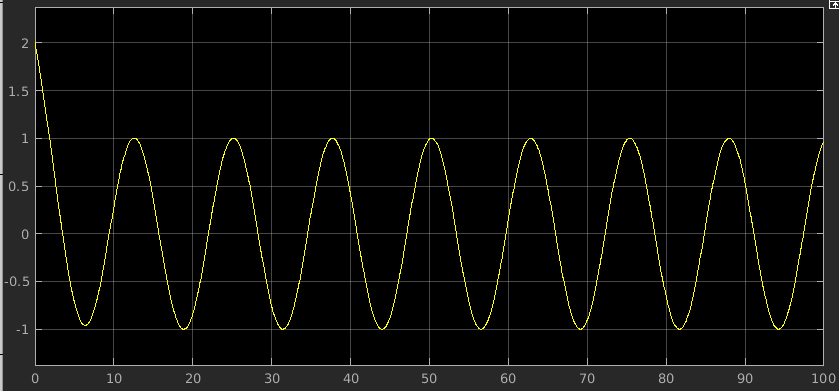


Рисунок 4 – График для структурной схемы

Можно заметить, что графики совпали, схема составлена верно.

**3)** Приведем в форме Вход-Состояние-Выход:

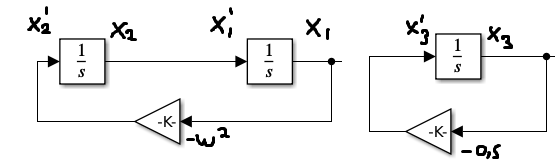
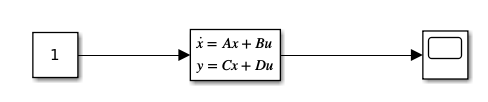


Рисунок 5



С помощью блока State-Space можно реализовать в Simulink:



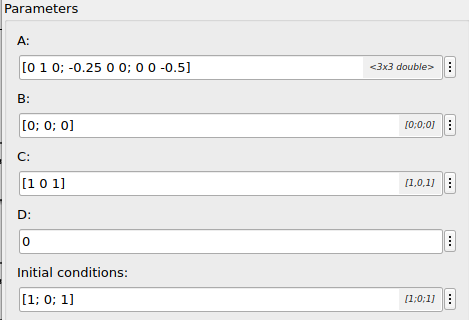
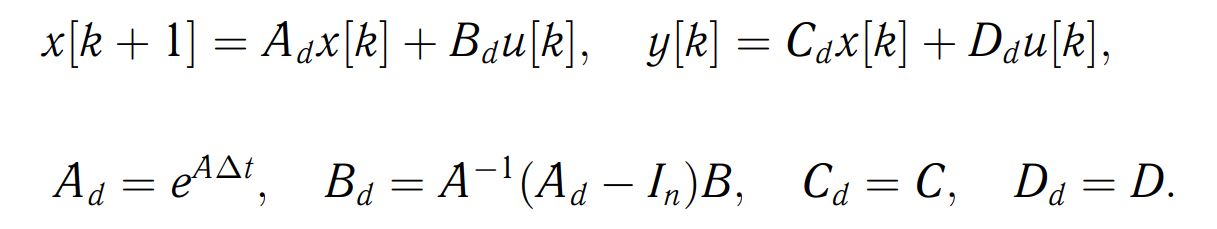
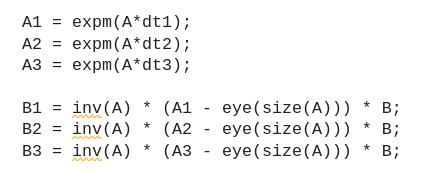


Рисунок 6

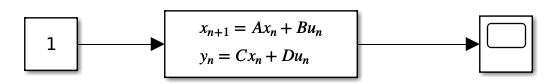
**4)** Дискретизируем генератор с частотой дискретизации 5, 30, 100 Гц.



Получить новые матрицы можно с помощью следующего скрипта:



Для реализации используется блок Discrete State-Space:



Получим следующие графики для непрерывного и дискретного вида (графики в сравнении с непрерывном приведены в **приложении 1**):

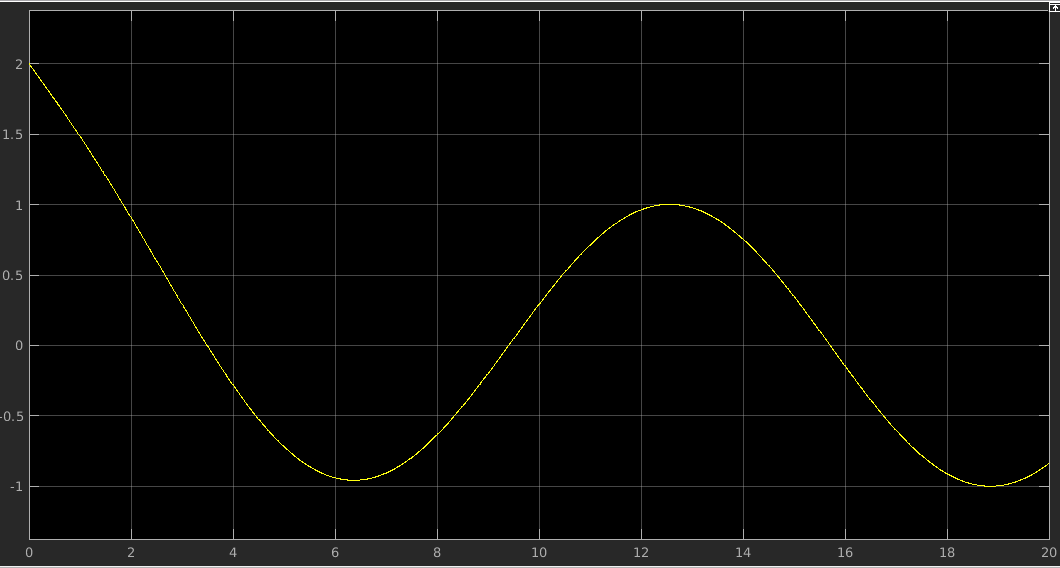


Рисунок 7 – Для непрерывной системы

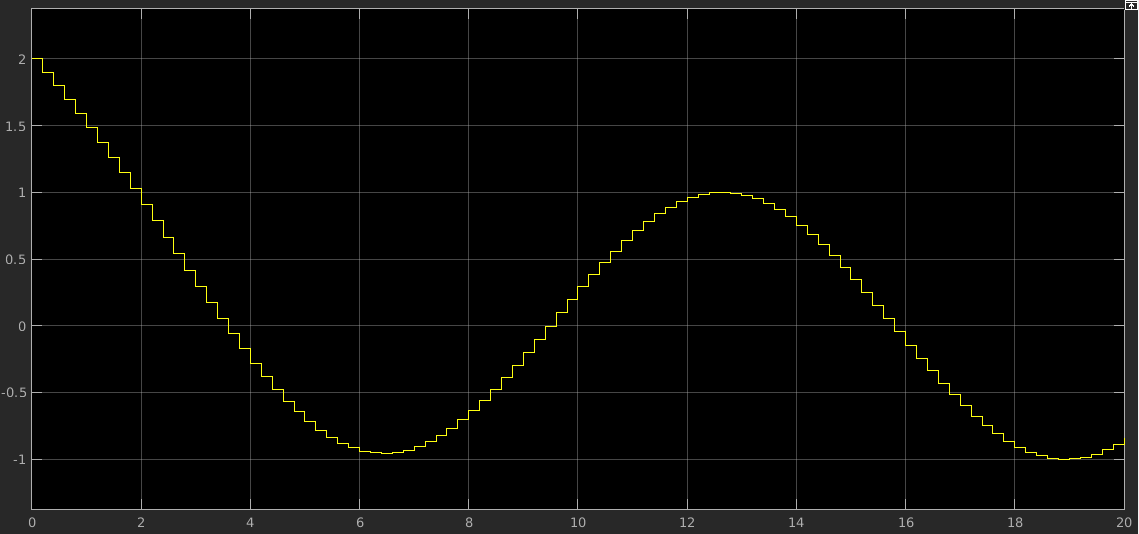


Рисунок 8 – Дискретный вид для частоты 5 Гц

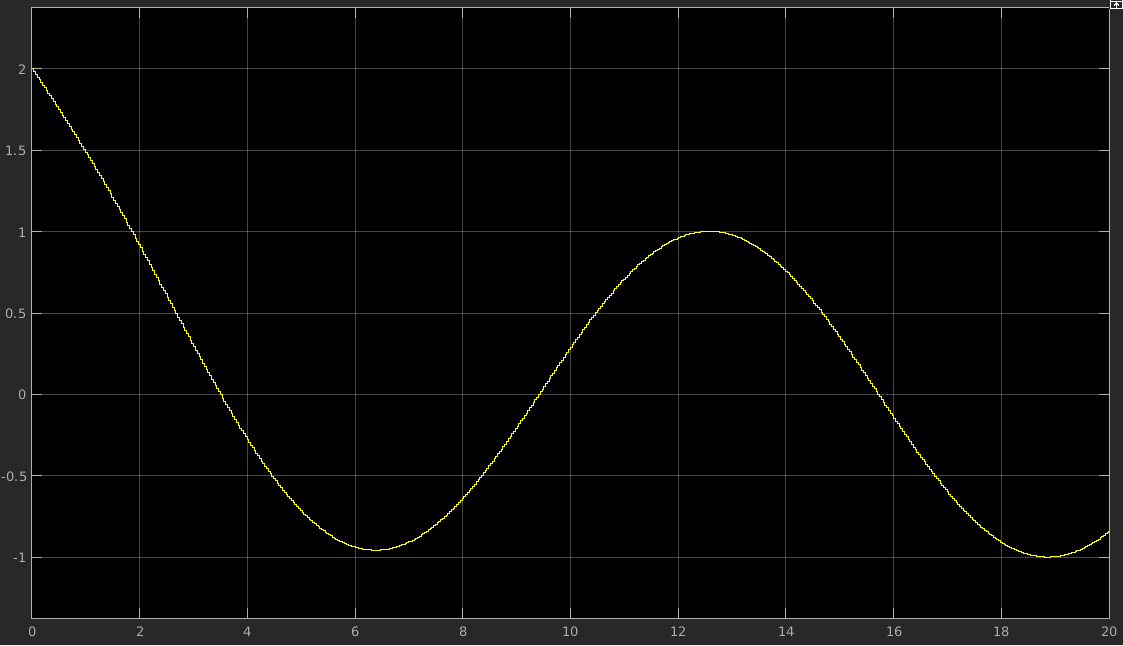


Рисунок 9 – Дискретный вид для частоты 30 Гц

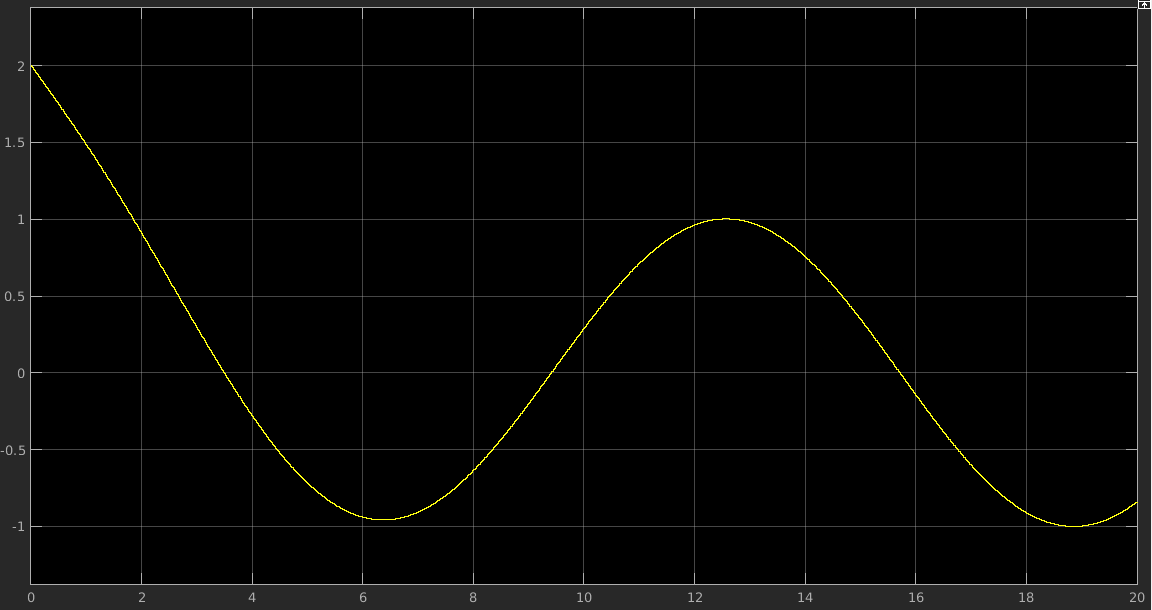


Рисунок 10 – Дискретный вид для частоты 30 Гц

Несложно заметить, что с увеличением частоты уменьшается шаг дискретизации, что делает график более гладким.

**5)** Имеем следующую передаточную функцию:

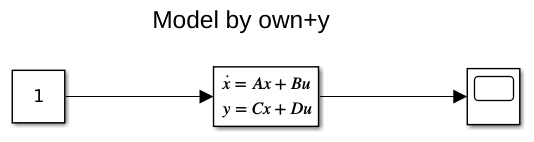


Получим вид Вход-Состояние-Выход:





С помощью блока State-Space в Simulink реализуем данную схему:



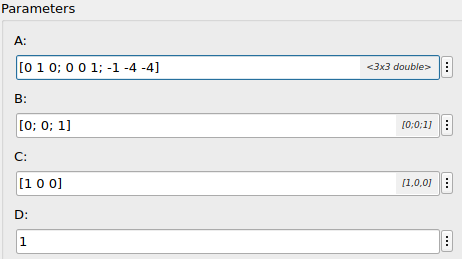


Рисунок 11 – Модель в виде ВСВ

Полученный вид - управляемая каноническая форма. Для сравнения, получим ее с помощью скрипта на matlab, а также получим наблюдаемую каноническую форму:

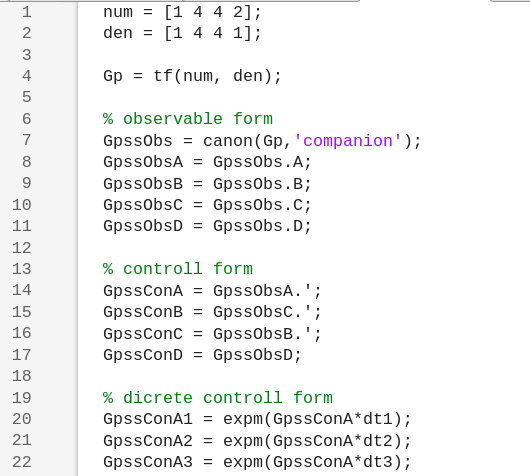


Рисунок 12

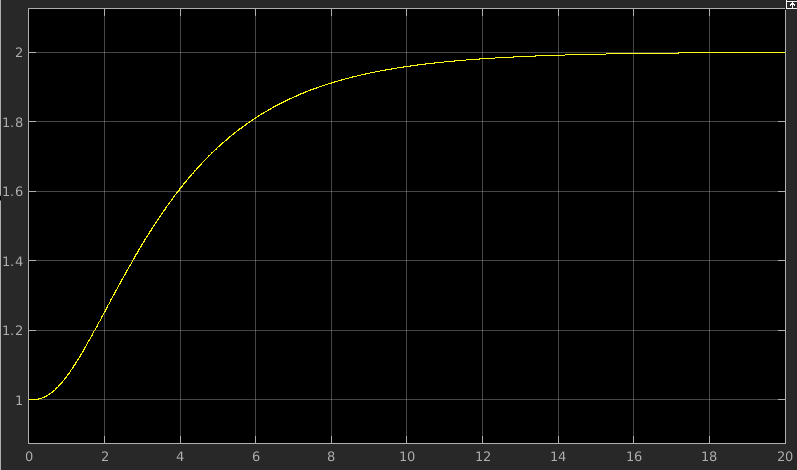


Рисунок 13 – ВСВ в управляемой канонической форме

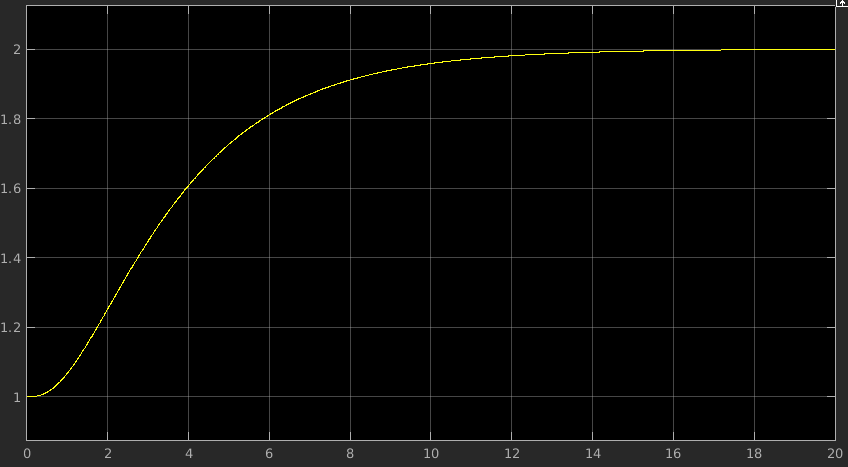


Рисунок 14 – ВСВ в наблюдаемой канонической форме

Видим, что графики совпадают. Также скрипт дает аналогичные матрицы для управляемой канонической формы, полученные вручную.

**6)** На основе модели ВСВ составим структурную схему:

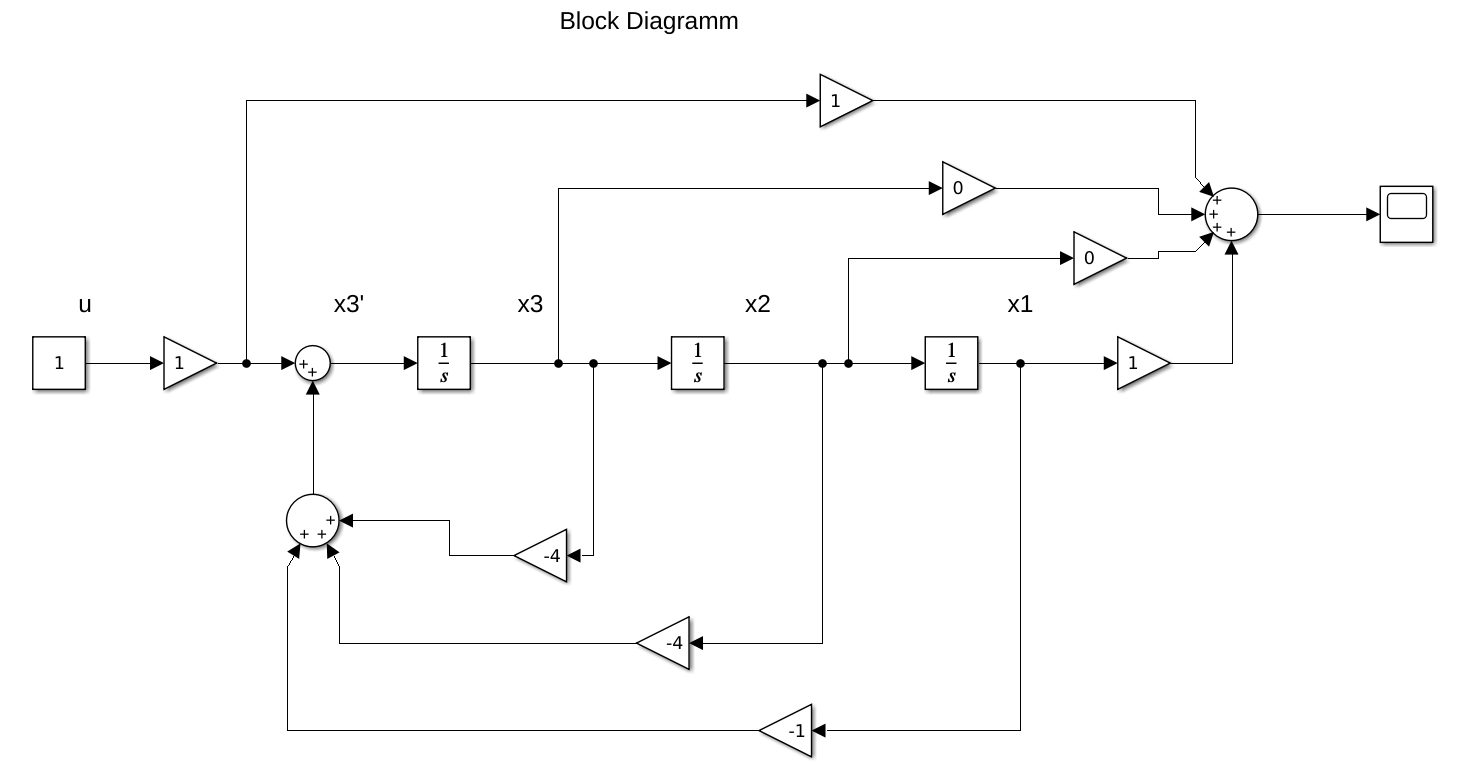


Рисунок 15 – Структурная схема объекта управления

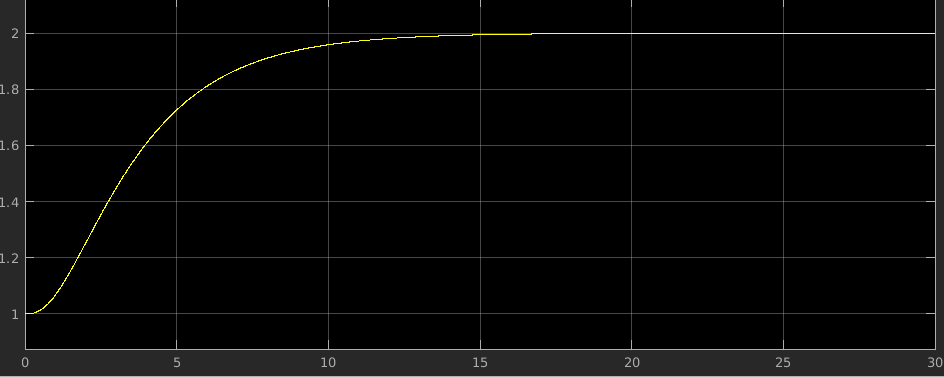


Рисунок 16 – График для структурной схемы

**7)** Дискретизация проводится по такому же принципу, как и для генератора. Получим следующие графики для частот 5, 30, 100 Гц.

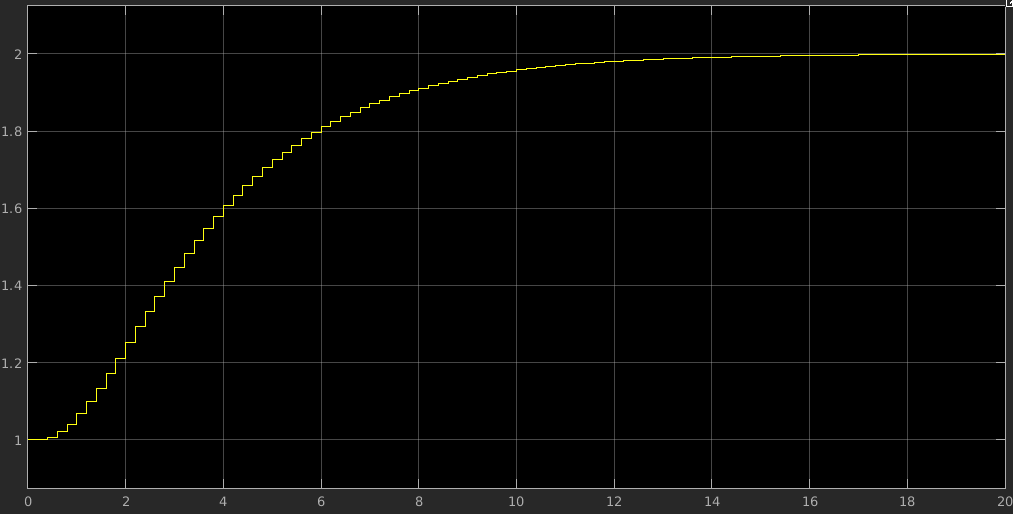


Рисунок 17 – Дискретная модель для частоты 5 Гц

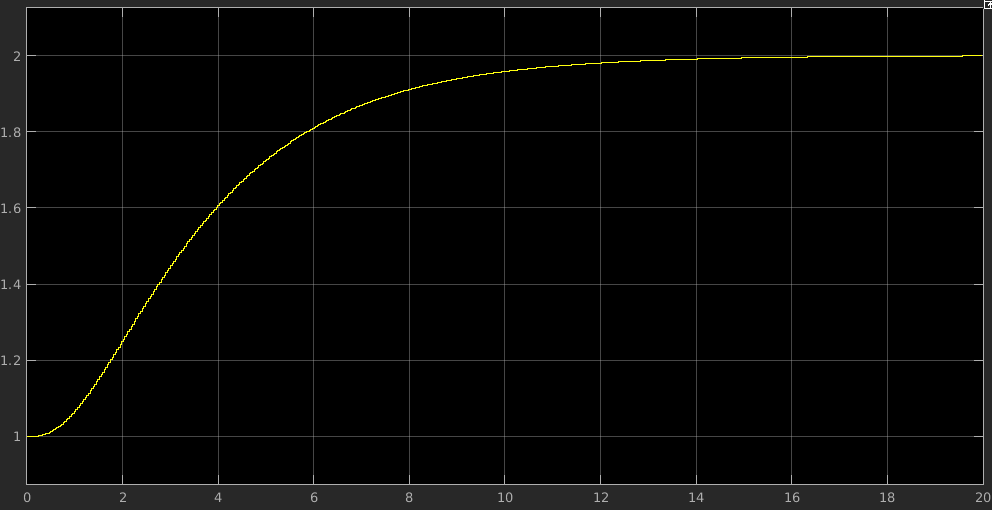


Рисунок 18 – Дискретная модель для частоты 30 Гц

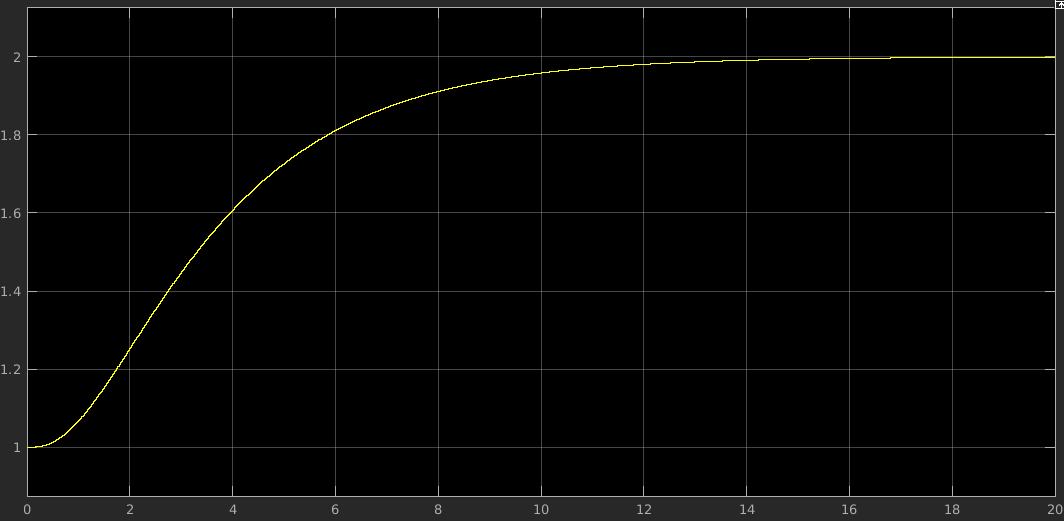


Рисунок 19 – Дискретная модель для частоты 100 Гц

Аналогично, с увеличением частоты уменьшается шаг, и график "сглаживается".

**8)** Программно реализуем класс интегратора. В качестве метода аппроксимации используется метод трапеции:

lintegrator.h

|  |
| --- |
| #include <QObject>  **class** LIntegrator : **public** QObject  {  Q\_OBJECT  **public**:  LIntegrator(**double** state, QObject \*parent = **nullptr**);  **double** update(**double** inputVal, **double** dt);  **double** getState();  **private**:  **double** state\_;  **double** prevState\_;  }; |

lintegrator.cpp

|  |
| --- |
| #include "lintegrator.h"  LIntegrator::LIntegrator(**double** state, QObject \*parent)  : QObject(parent), state\_(state), prevState\_(0.0)  {}  **double** LIntegrator::update(**double** inputVal, **double** dt)  {  **this**->state\_ = state\_ + (prevState\_ + inputVal) \* dt / 2.0;  // y[k-1] + (u[k] + u[k+1]) \* dt/2  // this->state\_ = state\_ + prevState\_ \* dt;  prevState\_ = inputVal;  **return** state\_;  }  **double** LIntegrator::getState()  {  **return** state\_;  } |

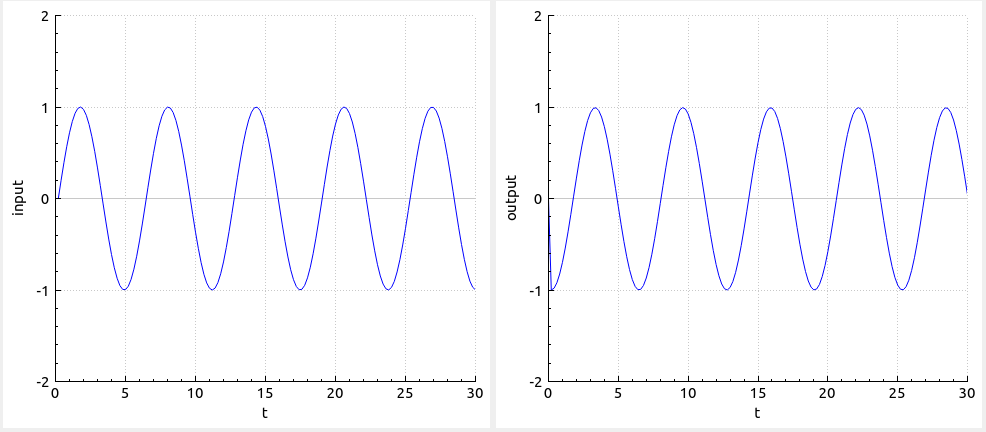


Рисунок 19 – Слева синусоида, справа - результат интегратора

**9)** Программно реализуем структурную схему генератора. Получим следующий класс:

|  |
| --- |
| **class** LIntegrator;  **class** Gain;  **class** GeneratorException{};  **class** Generator : **public** QObject  {  Q\_OBJECT  **public**:  Generator(std::vector<**double**>& states, std::vector<**double**>& gains, QObject \*parent = **nullptr**);  **double** update(**double** inputVal, **double** dt);  **private**:  **void** InitGenerator(**double** state1, **double** state2, **double** state3, **double** gain1, **double** gain2);  LIntegrator\* integrator\_1;  LIntegrator\* integrator\_2;  LIntegrator\* integrator\_3;  Gain\* gain\_1;  Gain\* gain\_2;  }; |

|  |
| --- |
| #include "generator.h"  #include "lintegrator.h"  #include "blocks/gain/gain.h"  Generator::Generator(std::vector<**double**> &states, std::vector<**double**> &gains, QObject \*parent)  : QObject(parent),  integrator\_1(**nullptr**), integrator\_2(**nullptr**), integrator\_3(**nullptr**),  gain\_1(**nullptr**), gain\_2(**nullptr**){  **if** (states.size() < 3 || gains.size() < 2) **throw** GeneratorException();  InitGenerator(states[0], states[1], states[2], gains[0], gains[1]);  }  **double** Generator::update(**double** inputVal, **double** dt)  {  **double** currState2 = **this**->integrator\_2->getState();  **double** resState2 = integrator\_2->update(  integrator\_1->update(gain\_1->update(currState2), dt),  dt);  **double** currState3 = integrator\_3->getState();  **double** resState3 = integrator\_3->update(gain\_2->update(currState3), dt);  **return** resState2 + resState3;  }  **void** Generator::InitGenerator(**double** state1, **double** state2, **double** state3, **double** gain1, **double** gain2)  {  integrator\_1 = **new** LIntegrator(state1);  integrator\_2 = **new** LIntegrator(state2);  integrator\_3 = **new** LIntegrator(state3);  gain\_1 = **new** Gain(gain1);  gain\_2 = **new** Gain(gain2);  } |

Для правильной реализации, необходимо запоминать текущее состояние интеграторов, передавая данные значения в качестве входа на умножитель (gain).

Получим следующий график для реализованного генератора:

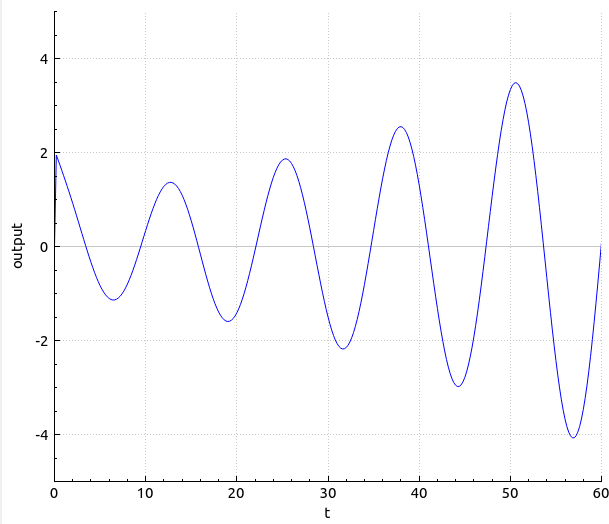


Рисунок 20 – График для генератора

Можно заметить, что со временем идет расхождение. Метод трапеций, примененный в интеграторе, неидеальный и имеет ошибку. Кроме того, интегратор, реализованный через структурную схему, накапливает ошибку. Ошибка от интегратора попадает на блок gain, после чего, идет на вход другого интегратора и т.д.

Для сравнения, ниже приведен график для метода прямоугольников. Можно увидеть, что данный метод обладает большей ошибкой.

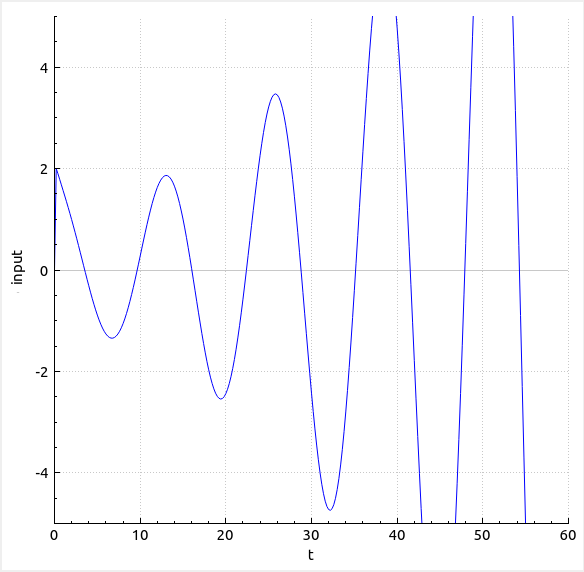


Рисунок 21 – Для метода прямоугольников

Расхождение структурной схемы можно также получить в системе Simulink, установив в интеграторах метод Эйлера:

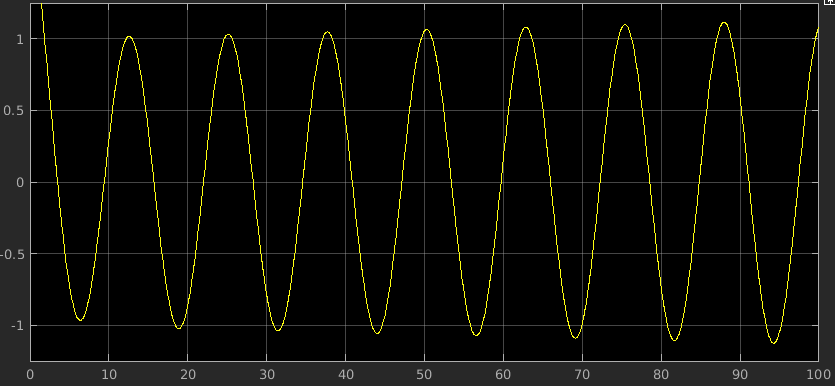
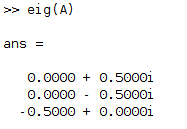


Рисунок 22 – График при использовании метода Эйлера



Дискретный генератор реализован с помощью класса, описанного ниже (вместе с дискретным объектом управления). По рисунку ниже видно, что дискретный генератор обладает устойчивостью, в отличие от непрерывного.

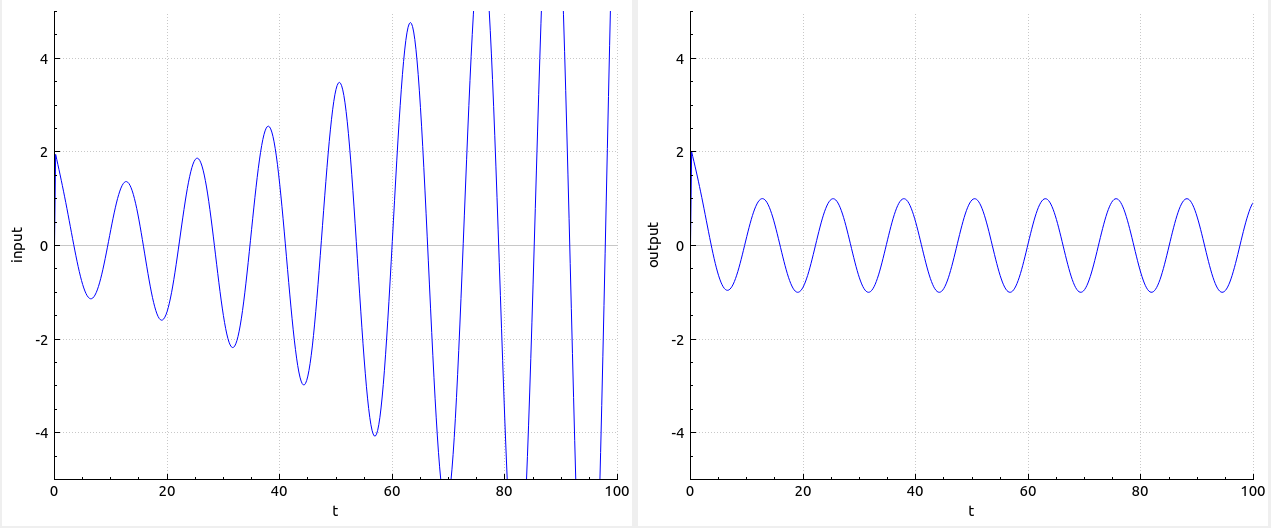


Рисунок 23 – Сравнение непрерывного (метод трапеций) и дискретного генераторов

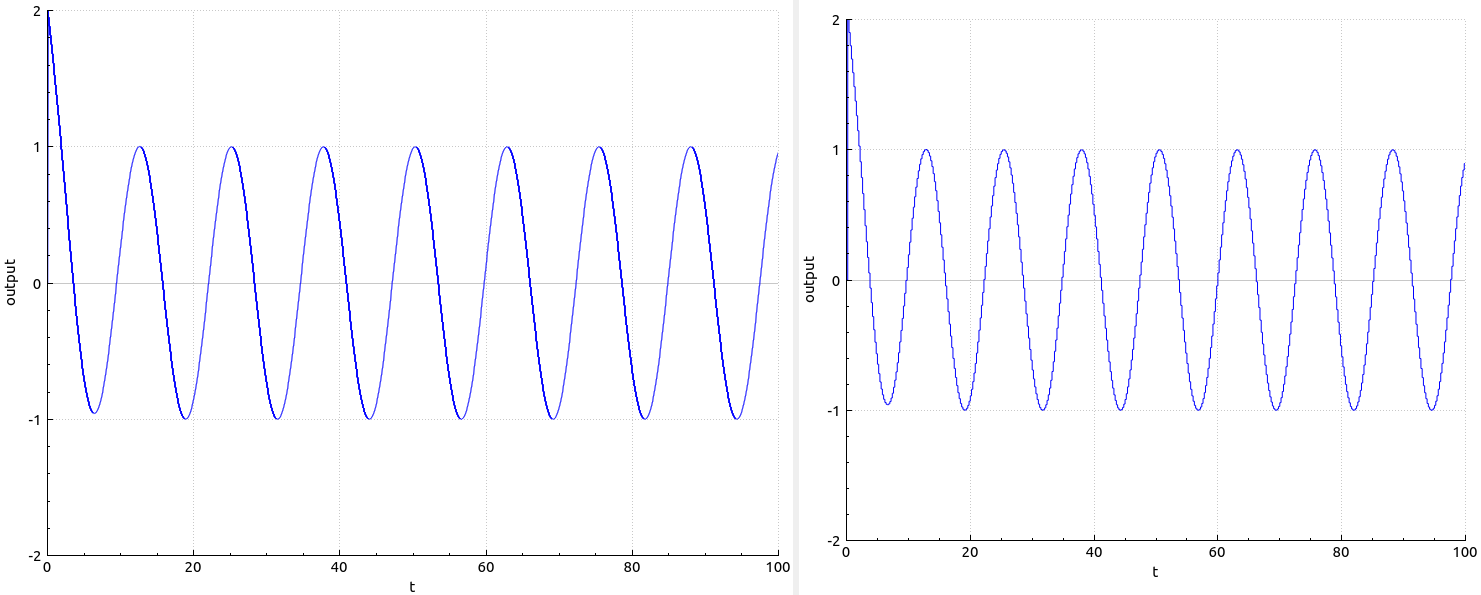
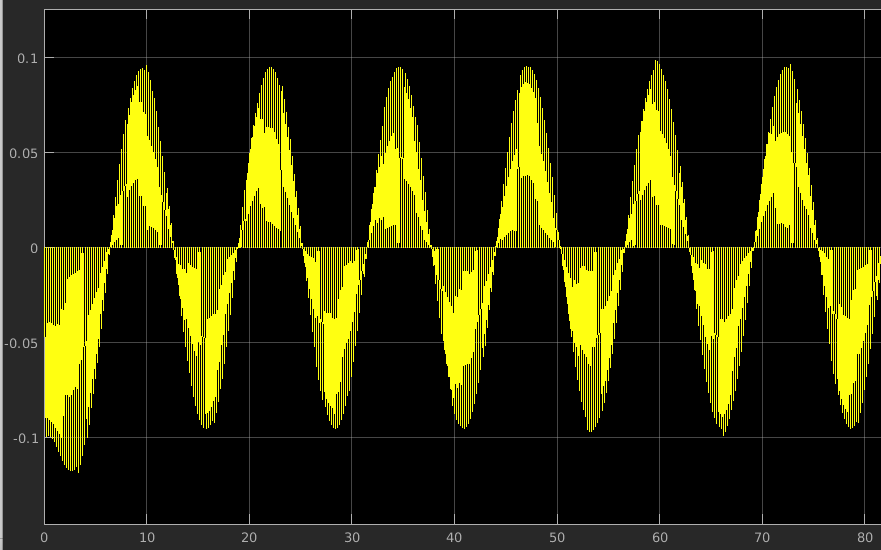
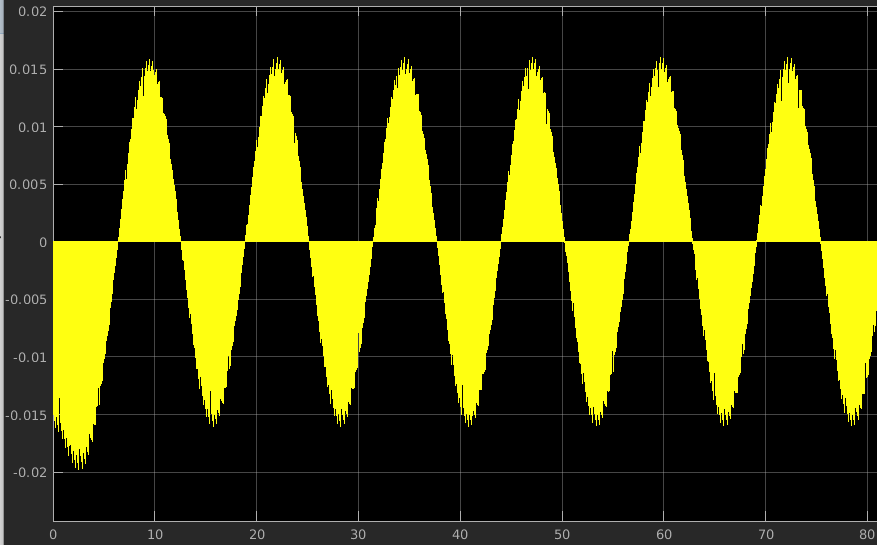
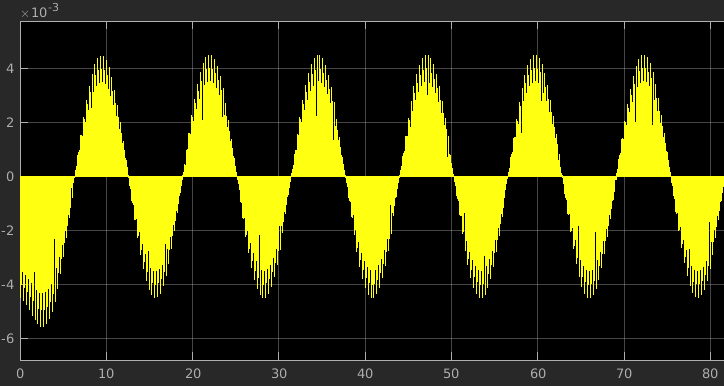


Рисунок 24 – Слева для частоты 30 Гц, справа для частоты 5 Гц

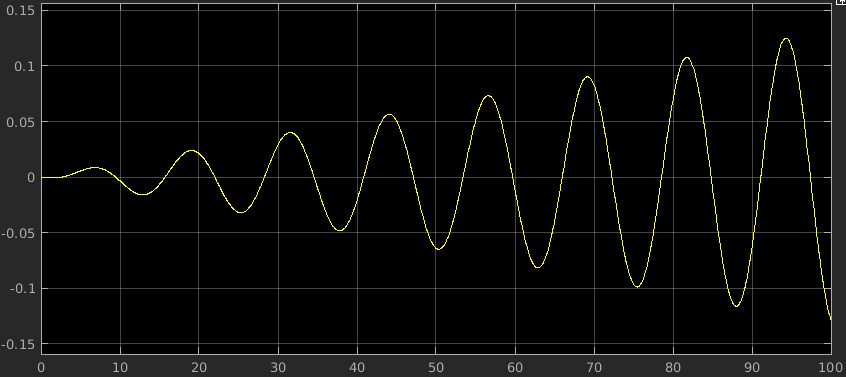
Частота дискретизации позволяет улучшить точность приближения, что можно увидеть, если рассмотреть графики разницы между значением, полученным по формуле (блок Matlab function) и реализованной дискретной формой:

Для частоты 5 Гц

Для частоты 30 Гц

Для частоты 100 Гц

Можно также рассмотреть поведение ошибки, при установке метода Эйлера для интеграторов:



Видно, что непрерывная модель расходится.

**10)** Программно реализуем объект управления. Получим следующий класс, представленный ниже. Данный класс принимает переменное число интеграторов и умножителей.

|  |
| --- |
| #include <QVector>  #include <QPair>  **class** LIntegrator;  **class** Gain;  **class** ControlModelError{};  **class** ControlModel  {  **public**:  ControlModel(QVector<**double**>& states, QVector<QPair<**double**, **double**>>& gains, QPair<**double**, **double**>& sigGain);  **double** update(**double** inputVal, **double** dt);  **private**:  QVector<LIntegrator\*> integrators\_;  QVector<QPair<Gain\*,Gain\*>> gains\_;  QPair<Gain\*,Gain\*> inputSignalGain\_;  }; |

|  |
| --- |
| #include "controlmodel.h"  #include "lintegrator.h"  #include "blocks/gain/gain.h"  /\*\* setting integrators from left to right      gains set from first = by init state to second = result state  \*/  ControlModel::ControlModel(QVector<**double**>& states, QVector<QPair<**double**, **double**>>& gains,  QPair<**double**, **double**>& sigGain)  {  **if** (states.size() == 0 || states.size() != gains.size())  **throw** ControlModelError();  **for** (**auto** state : states){  integrators\_.push\_back(**new** LIntegrator(state));  }  **for** (**auto** currgain: gains){  gains\_.push\_back(QPair<Gain\*, Gain\*>(  **new** Gain(currgain.first), **new** Gain(currgain.second)));  }  **this**->inputSignalGain\_.first = **new** Gain(sigGain.first);  **this**->inputSignalGain\_.second = **new** Gain(sigGain.second);  }  **double** ControlModel::update(**double** inputVal, **double** dt)  {  **double** currStates[integrators\_.size()];  **for** (**int** i = integrators\_.size()-1; i >= 0; i--)  {  currStates[i] = gains\_[i].first->update(integrators\_[i]->getState());  }  **double** currInput = inputSignalGain\_.first->update(inputVal);  /\*\* init for result summ \*/  **double** currInputIntegrator = currInput;  **for** (**int** i = 0; i < integrators\_.size(); i++){  currInputIntegrator += currStates[i];  }  **for** (**int** i = 0; i < integrators\_.size(); i++){  currInputIntegrator = integrators\_[i]->update(currInputIntegrator, dt);  }  /\*\* get result summ \*/  currInput = inputSignalGain\_.second->update(currInput);  **for** (**int** i = 0; i < integrators\_.size(); i++){  currInput += gains\_[i].second->update(integrators\_[i]->getState());  }  **return** currInput;  } |

Получим следующий график:

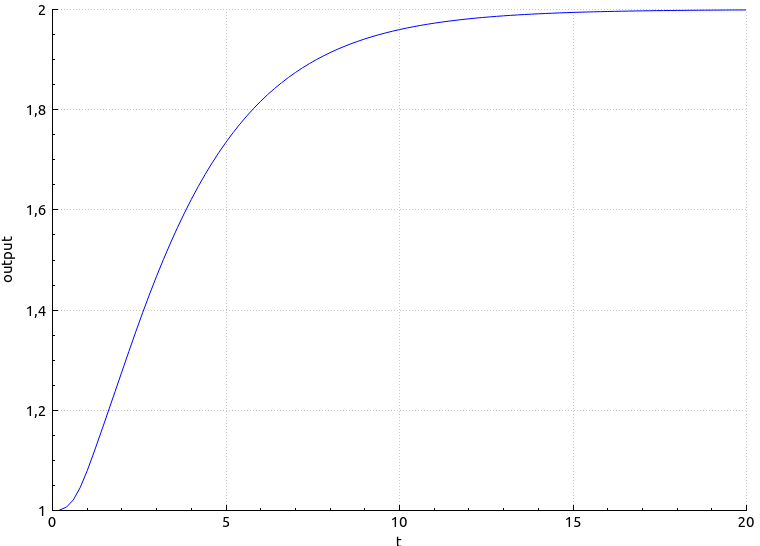


Рисунок 25

**11)** Программно реализуем дискретные модели объектов управления с помощью реккурентных формул:



Был реализован простой класс матрицы:

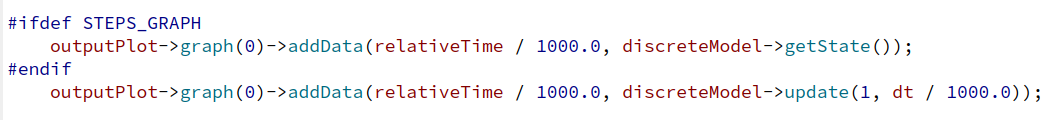
|  |
| --- |
| **class** SimpleMatrix{  **public**:  SimpleMatrix(size\_t rows, size\_t columns, **double** initVal=0.0);  **double** getElem(size\_t row, size\_t col);  **void** setElem(size\_t row, size\_t col, **double** value);  **void** getSize(**int**& row, **int**& col);  size\_t getRows();  size\_t getColumns();  **private**:  QVector<QVector<**double**>> data\_;  size\_t rows\_;  size\_t columns\_;  }; |

|  |
| --- |
| **class** DiscreteControlModelError{};  **class** DiscreteControlModel  {  **public**:  DiscreteControlModel(SimpleMatrix\* A, QVector<**double**> B, QVector<**double**> C, **double** D, QVector<**double**> x, **double** y);  **double** update(**double** val, **double** dt);  **double** getState();  ~DiscreteControlModel();  **private**:  SimpleMatrix\* A\_;  QVector<**double**> B\_;  QVector<**double**> C\_;  **double** D\_;  QVector<**double**> xCurr;  **double** yCurr;  }; |

|  |
| --- |
| #include <iostream>  SimpleMatrix::SimpleMatrix(size\_t rows, size\_t columns, **double** initVal)  : rows\_(rows), columns\_(columns)  {  QVector<**double**> currRow = QVector<**double**>(columns\_, initVal);  **for** (**int** i = 0; i < rows\_; i++){  data\_.push\_back(currRow);  }  }  **double** SimpleMatrix::getElem(size\_t row, size\_t col)  {  **return** data\_[row][col];  }  **void** SimpleMatrix::setElem(size\_t row, size\_t col, **double** value)  {  data\_[row][col] = value;  }  **void** SimpleMatrix::getSize(**int** &row, **int** &col)  {  row = rows\_; col = columns\_;  }  size\_t SimpleMatrix::getRows()  {  **return** rows\_;  }  size\_t SimpleMatrix::getColumns()  {  **return** columns\_;  }  DiscreteControlModel::DiscreteControlModel(SimpleMatrix\* A, QVector<**double**> B, QVector<**double**> C, **double** D, QVector<**double**> x, **double** y)  : A\_(A), B\_(B), C\_(C), D\_(D), xCurr(x), yCurr(y)  {  **if** (A\_->getColumns() != A\_->getRows() ||  A\_->getRows() != xCurr.size() ||  A\_->getRows() != B\_.size() ||  C\_.size() != xCurr.size())  **throw** DiscreteControlModelError();  }  **double** DiscreteControlModel::update(**double** val, **double** dt)  {  **auto** currX = xCurr;  /\*\* recalc x[k+1] = A\*x[k] + B\*u(t) \*/  **for** (**int** cRow = 0; cRow < xCurr.size(); cRow++)  {  xCurr[cRow] = B\_[cRow] \* val;  **for** (**int** cCol = 0; cCol < A\_->getColumns(); cCol++)  {  xCurr[cRow] += A\_->getElem(cRow, cCol) \* currX[cCol];  }  }  /\*\* recalc y by x[k] and return \*/  **auto** currY = yCurr;  yCurr = D\_ \* val;  **for** (**int** i = 0; i < C\_.size(); i++){  yCurr += C\_[i] \* currX[i];  }  **return** yCurr;  }  **double** DiscreteControlModel::getState()  {  **return** yCurr;  }  DiscreteControlModel::~DiscreteControlModel()  {  **delete** A\_;  } |

Получим следующие графики в сравнении с непрерывным видом объекта управления.

Ступенчатый вид графика можно легко получить следующим способом:



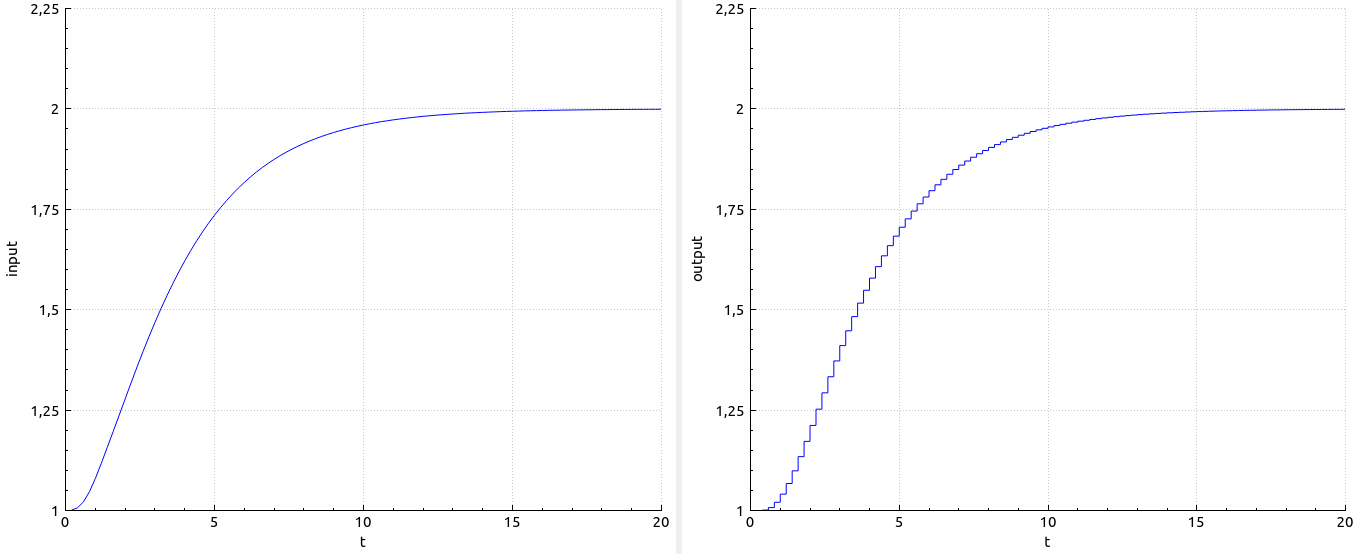


Рисунок 26 - Сравнение с дискретным объектом (5 Гц)

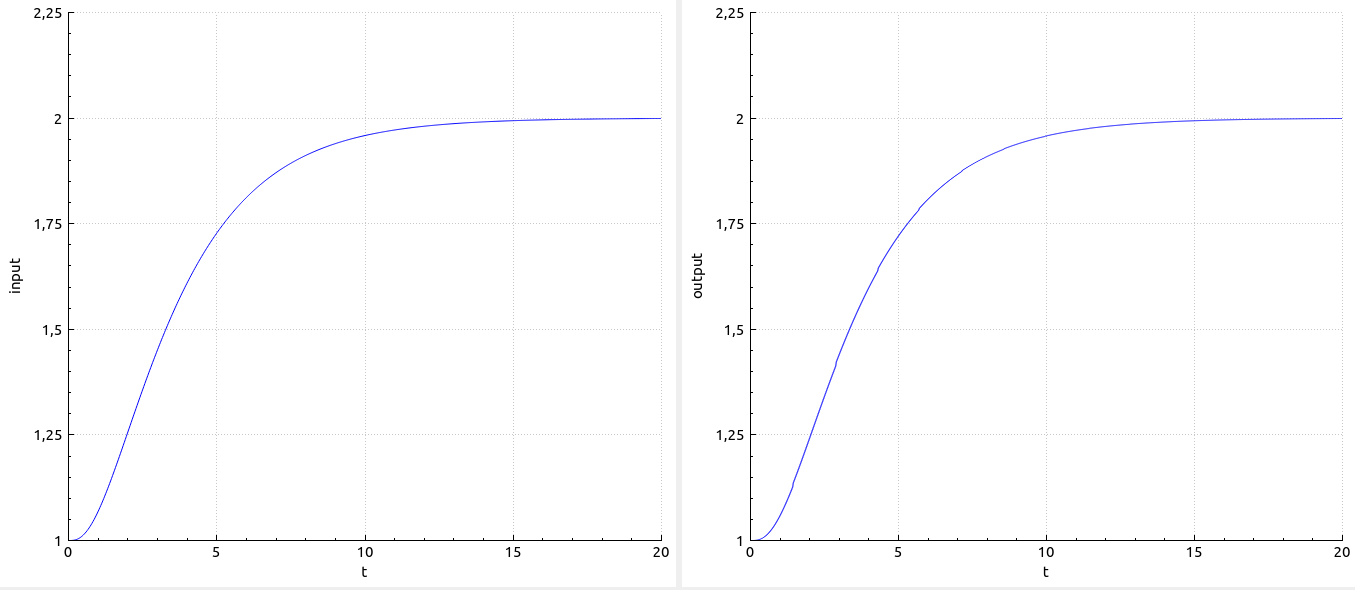


Рисунок 26 - Сравнение с дискретным объектом (30 Гц)

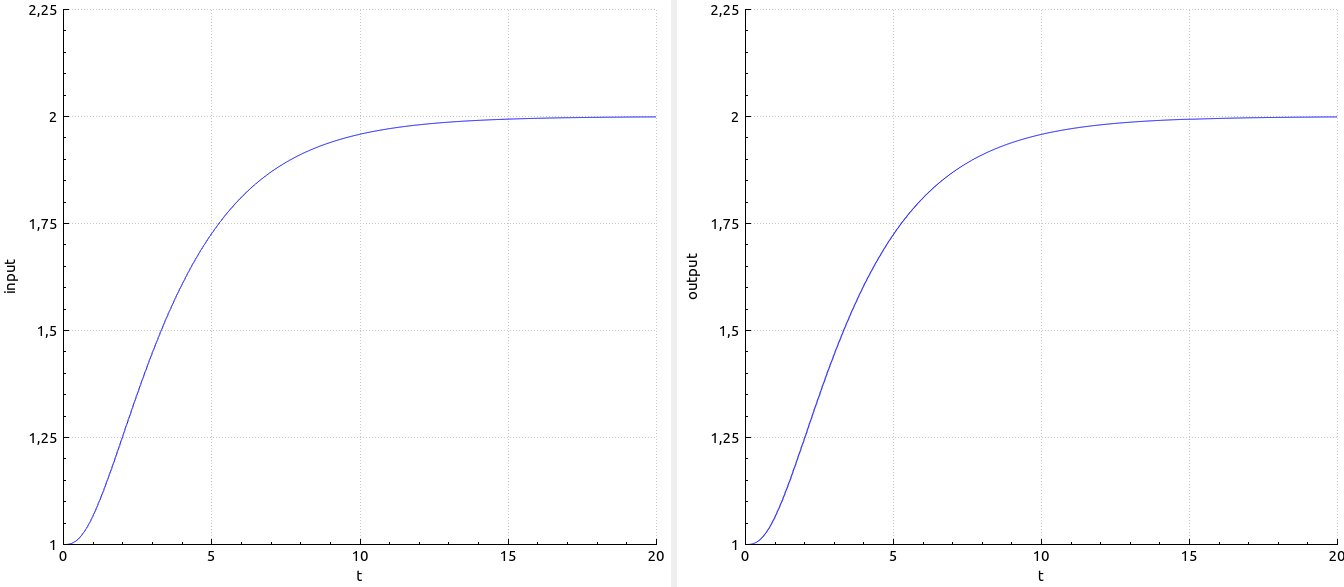
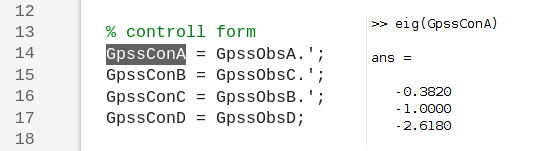


Рисунок 26 - Сравнение с дискретным объектом (100 Гц)

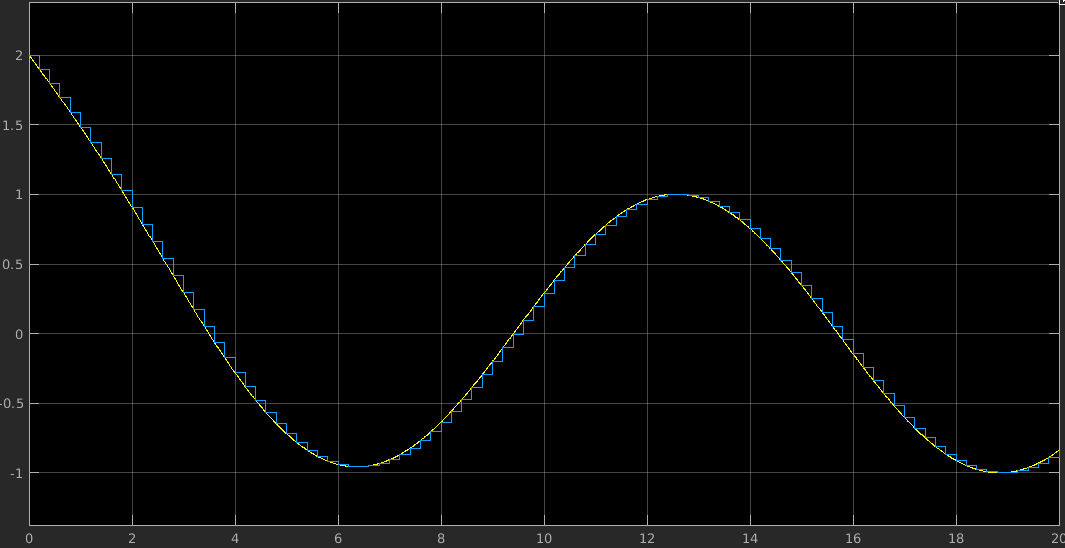
Аналогично можно увидеть, что увеличение частоты дискретизации, которое приводит к уменьшению шага, повышает гладкость графика. Непрерывный вид является устойчивым, это можно проверить, посмотрев на собственные числа матрицы A.



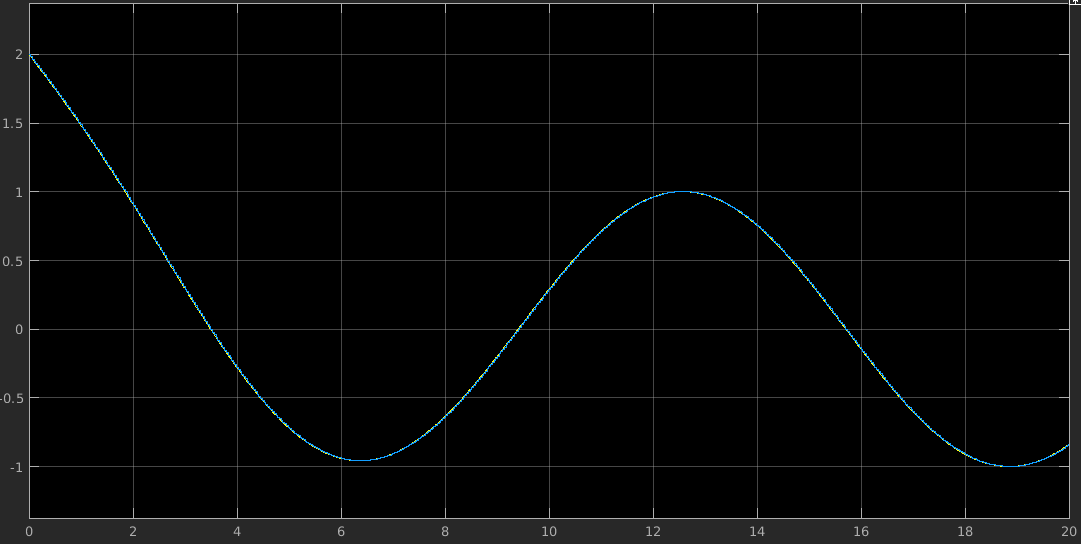
В ходе выполнения работы был реализован генератор в среде Simulink. Была реализована структурная схема для генератора, получена форма Вход-Состояние-Выход. Были рассчитаны матрицы и получены дискретные версии для частот дискретизации 5, 30 и 100 Гц. Для программной реализации в среде Qt Creator был реализован класс интегратора и на его основе реализована непрерывная версия генератора. Было проведено сравнение, в результате которого можно было наблюдать, что с увеличением частоты уменьшается шаг дискретизации, график становится более гладким, а точность апроксимации увеличивается. Т.к. интегратор был реализован с помощью метода трапеций, то при моделировании видно, что непрерывный генератор расходится. Метод трапеций имеет ошибку, которая увеличивается на блоке gain и подается на вход другому интегратору.

В среде Simulink был реализован объект управления. Была получена управляемая каноническая форма, структурная схема и дискретные формы. Аналогично, были реализованы соответствующие классы на языке С++.

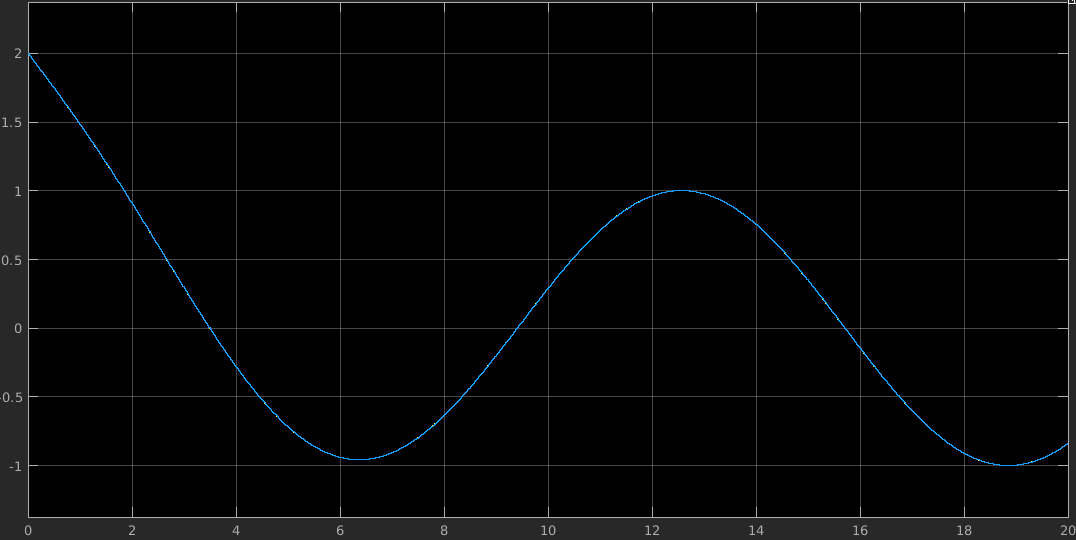
**Приложение 1**



Генератор, 5 ГЦ и непрерывная



Генератор, 30 ГЦ и непрерывная



Генератор, 100 ГЦ и непрерывная

Приложение 2

Полный код проекта и matlab моделей и скриптов можно посмотреть по ссылке.

<https://github.com/IlyaFedorov115/Programming_Control_System_ITMO/tree/Matlab_Part>

<https://github.com/IlyaFedorov115/Programming_Control_System_ITMO/tree/Qt_part>

Код для матриц (как константы).

|  |
| --- |
| #ifndef CONSTANS\_H  #define CONSTANS\_H  #include <QVector>  #include "discretecontrolmodel.h"  /\*\* Matrixes for Discrete Generator \*/  SimpleMatrix\* getDGeneratorA1()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.99500417); result->setElem(0, 1, 0.19966683); result->setElem(0, 2, 0.00000000);  result->setElem(1, 0, -0.04991671); result->setElem(1, 1, 0.99500417); result->setElem(1, 2, 0.00000000);  result->setElem(2, 0, 0.00000000); result->setElem(2, 1, 0.00000000); result->setElem(2, 2, 0.90483742);  **return** result;  }  SimpleMatrix\* getDGeneratorA2()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.99986388); result->setElem(0, 1, 0.03299850); result->setElem(0, 2, 0.00000000);  result->setElem(1, 0, -0.00824963); result->setElem(1, 1, 0.99986388); result->setElem(1, 2, 0.00000000);  result->setElem(2, 0, 0.00000000); result->setElem(2, 1, 0.00000000); result->setElem(2, 2, 0.98363538);  **return** result;  }  SimpleMatrix\* getDGeneratorA3()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.99998750); result->setElem(0, 1, 0.00999996); result->setElem(0, 2, 0.00000000);  result->setElem(1, 0, -0.00249999); result->setElem(1, 1, 0.99998750); result->setElem(1, 2, 0.00000000);  result->setElem(2, 0, 0.00000000); result->setElem(2, 1, 0.00000000); result->setElem(2, 2, 0.99501248);  **return** result;  }  QVector<**double**> DGeneratorB {0.0, 0.0, 0.0};  QVector<**double**> DGeneratorC {1.0, 0.0, 1.0};  **double** DGeneratorD = 0.0;  /\*\* Matrixes For Discrete Control object \*/  SimpleMatrix\* getAdt1()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.998904057); result->setElem(0, 1, 0.195559240); result->setElem(0, 2, 0.015385936);  result->setElem(1, 0, -0.015385936); result->setElem(1, 1, 0.937360312); result->setElem(1, 2, 0.134015495);  result->setElem(2, 0, -0.134015495); result->setElem(2, 1, -0.551447918); result->setElem(2, 2, 0.401298331);  **return** result;  }  SimpleMatrix\* getAdt2()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.999994204); result->setElem(0, 1, 0.032976769); result->setElem(0, 2, 0.000521124);  result->setElem(1, 0, -0.000521124); result->setElem(1, 1, 0.997909707); result->setElem(1, 2, 0.030892272);  result->setElem(2, 0, -0.030892272); result->setElem(2, 1, -0.124090211); result->setElem(2, 2, 0.874340620);  **return** result;  }  SimpleMatrix\* getAdt3()  {  SimpleMatrix\* result = **new** SimpleMatrix(3, 3);  result->setElem(0, 0, 0.999999835); result->setElem(0, 1, 0.009999340); result->setElem(0, 2, 0.000049338);  result->setElem(1, 0, -0.000049338); result->setElem(1, 1, 0.999802482); result->setElem(1, 2, 0.009801986);  result->setElem(2, 0, -0.009801986); result->setElem(2, 1, -0.039257284); result->setElem(2, 2, 0.960594536);  **return** result;  }  /\*\* For ALL \*/  QVector<**double**> GpssConC {1.0, 0.0, 0.0};  **double** GpssConD = 1.0;  /\*\* dt-1 \*/  QVector<**double**> GpssConB1 {0.001095943, 0.015385936, 0.134015495};  /\*\* dt-2 \*/  QVector<**double**> GpssConB2 {0.000005796, 0.000521124, 0.030892272};  /\*\* dt-3 \*/  QVector<**double**> GpssConB3 {0.000000165, 0.000049338, 0.009801986};  #endif // CONSTANS\_H |