ДК 004.942

Рыбалёв Андрей Николаевич
Амурский государственный университет
г. Благовещенск, Россия
E-mail: amgu_appe@mail.ru
Rybalev Andrey Nikolaevich
Amur State University
Blagoveshchensk, Russia
E-mail: amgu_appe@mail.ru

ФОРМИРОВАНИЕ УПРАВЛЯЮЩИХ ПРОГРАММ ДЛЯ СИСТЕМ АВТОМАТИЧЕСКОГО РЕГУЛИРОВАНИЯ НА ОСНОВЕ ЧИСЛЕННОГО ИНТЕГРИРОВАНИЯ И ТЕХНОЛОГИИ ООП

FORMATION OF CONTROL PROGRAMS FOR AUTOMATIC REGULATION SYSTEMS BASED ON NUMERICAL INTEGRATION AND OOP TECHNOLOGY

Аннотация. Предложен подход к программной реализации динамических структур для ПЛК с привлечением методов численного интегрирования и элементов объектно-ориентированного программирования.

Abstract. An approach to the software implementation of dynamic structures for PLCs with the use of numerical integration methods and elements of object-oriented programming is proposed.

Ключевые слова: программируемый логический контроллер, численное интегрирование, интерфейс, полиморфизм.

Key words: programmable logic controller, numerical integration, interface, polymorphism.

Введение

Разработка программ автоматического регулирования для ПЛК на основе современных алгоритмов управления является в настоящее время актуальной и нетривиальной задачей. С одной стороны, случаи применения этих алгоритмов на практике ограничены специальными проектами, а в большинстве промышленных систем задействуются простейшие законы управления (релейный, ПИД-регулирование и т.д.). С другой стороны, отсутствуют общепринятые подходы переноса сложной «математики» в программный код для ПЛК.

Продвинутые алгоритмы требуют апробации и настройки на компьютерных моделях. В мире персональных компьютеров эта задача решается в системах имитационного моделирования класса Simulink Matlab. В результате создается графическая модель, с помощью которой можно отследить изменение любого сигнала, настроить коэффициенты и т.д. В принципе, полученную конструкцию можно почти один к одному перенести в ПЛК, воссоздав ее

средствами графических языков программирования стандарта МЭК 61131. Однако этот подход имеет определенные недостатки:

1. Трудоемкость. Simulink-модель алгоритма адаптивного управления может состоять из нескольких подсистем, каждая из которых содержит более десятка блоков. К этому нужно прибавить блоки, входящие в состав модели объекта, их тоже желательно перенести в программу контроллера, – хотя бы и временно, для отладки. Возможно, что в некоторых случаях даже на этапе имитационного моделирования имеет смысл отказаться от графического представления, вернувшись к написанию функций и скриптов, поскольку алгоритм намного проще конвертировать в код на текстовом языке.

2. Погрешности переноса. Реализация математических операций в системах имитационного моделирования, заточенных на «математику», зачастую существенно отличается от таковой в системах программирования ПЛК, ориентированных на «простоту реализации». Речь идет как о типах данных, так и об алгоритмах обработки/преобразования сигналов, в частности интегрирования. Частично эта проблема была затронута в [1].

В целом можно сказать, что существует потребность в разработке некоторого «шаблона», позволяющего быстро обращать математические конструкции, лежащие в основе как алгоритма управления, так и модели объекта, в код на текстовом языке, и, что еще более важно, – в систему исполнения этого кода на программируемых логических контроллерах. Этому вопросу и посвящена данная работа. Систему исполнения предлагается строить на хорошо известных алгоритмах численного интегрирования, а универсальность подхода может быть достигнута путем применения технологий объектно-ориентированного программирования, которые постепенно проникают в мир ПЛК [2].

Исследования проводились в Softlogic-системе CODESYS 3.5.

Традиционный подход

В качестве примера рассмотрим простейшую одноконтурную систему автоматического регулирования. Объект управления описывается передаточной функцией третьего порядка:

$$W_o(s) = \frac{8}{s^3 + 6s^2 + 12s + 8}.$$

Регулятор реализует пропорционально-итегральный закон регулирования:

$$W_p(s) = 1{,}14 + 0{,}908\frac{1}{s}.$$

Параметры регулятора найдены в Matlab с помощью функции pidtune пакета Control.

Программа регулятора и программная модель объекта показаны на рис. 1 и 2 соответственно. Они составлены на языке непрерывных функциональных схем CFC (Continuous Flow Chart) и на первый взгляд мало чем отличаются от соответствующих фрагментов Simulink-диаграммы системы регулирования, построенной по ее модели в пространстве состояний.

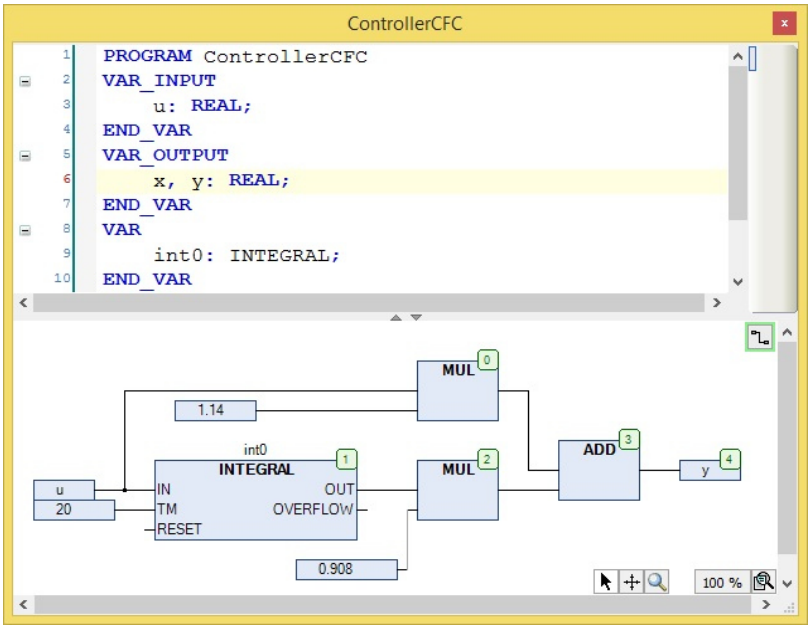

Рис. 1. Программа регулятора.

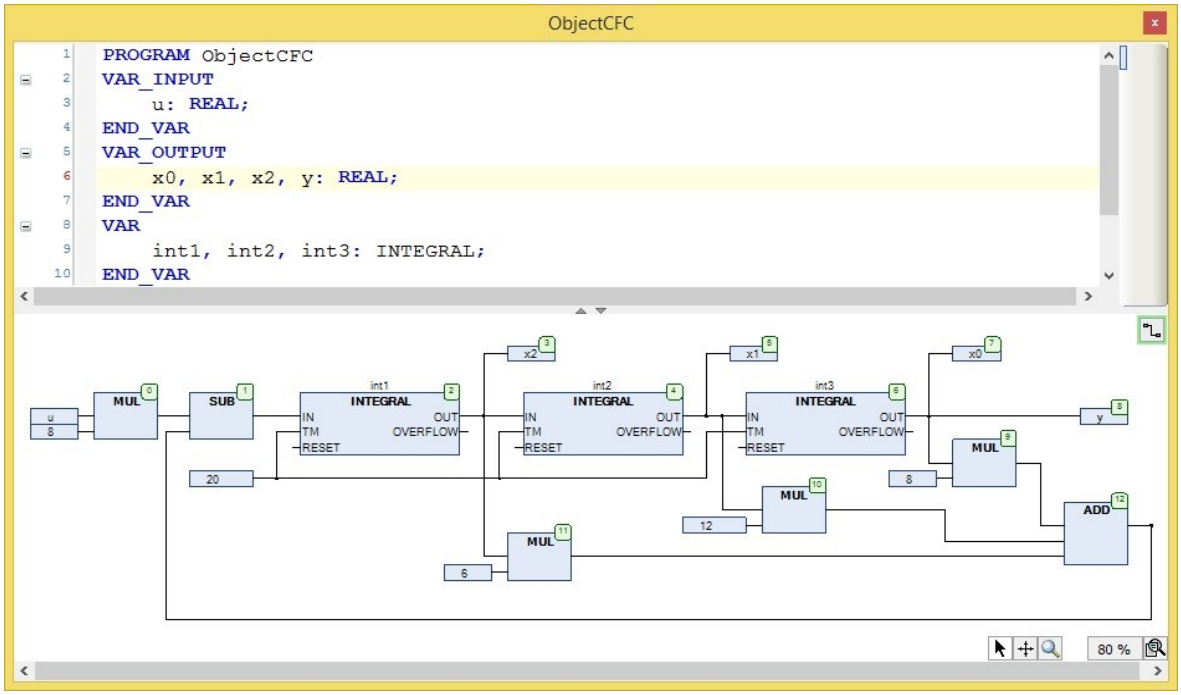

Рис. 2. Программная модель объекта управления.

Для «чистоты эксперимента» регулятор был намеренно построен без привлечения соответствующих библиотечных средств, имеющихся в системе, поскольку они скрывают детали реализации своего алгоритма. В программах задействуются экземпляры функционального блока INTEGRAL, выполняющего интегрирование методом трапеций. Интервал между вызовами – 20 мс.

Программы по очереди вызываются из головной программы PLC_PRG, при этом выход регулятора поступает на вход объекта, а выход объекта, совместно с сигналом задания, участвует в формировании входа регулятора:

ControllerCFC(u:=sp-yy,y=>uu);

ObjectCFC(u:=uu,y=>yy);

Сама программа PLC_PRG вызывается в рамках циклически выполняемой задачи (с периодом 20 мс).

Рассмотренный подход вне зависимости от языка реализации имеет один серьезный недостаток. Поскольку блоки пересчитываются *последовательно*, только один, первый в цепочке, интегратор (регулятора) будет на своем входе иметь значение, актуальное на момент начала периода. Второй интегратор (первый интегратор объекта) будет оперировать уже «новыми» данными, третий – «свехновыми» и т.д. В аналоговой же системе, которую мы считаем эталоном, все происходит «одновременно», т.е. все интеграторы всегда имеют на своих входах сигналы с одной и той же «меткой времени». При этом очевидно, что изменение структуры модели не решает проблему: в любом случае программа должна различать «новые» значения переменных состояния системы от «старых».

В Simulink используется совершенно другой подход. Перед расчетом графическая модель транслируется в набор дифференциальных уравнений первого порядка в форме Коши, которая потом решается одним из методов численного интегрирования. В результате само понятие *порядка пересчета* в Simulink применимо только к алгебраическим операциям (поэтому система и не приветствует «алгебраические циклы»), а интегрирование производится «параллельно».

Численное интегрирование в ПЛК

Сама по себе задача организации численного интегрирования средствами языков стандарта МЭК 61131 для конкретной системы не вызывает особых затруднений. В [1], в частности, приведен пример программы, в которой задействован метод Хойна (двухэтапный метод Рунге-Кутты второго порядка, [3]). Актуальной задачей представляется разработка общего подхода, который позволил бы на прикладном уровне использовать готовые решения с той же степенью простоты и свободы, с какой это делается в математических пакетах.

В текстовом языке Matlab для решения задачи привлекаются три программные сущности:

1) функция (ODEFUN), вычисляющая вектор производных переменных состояния системы (или, что то же самое, правые части дифференциальных уравнений). Эта функция создается пользователем и имеет фиксированный набор входных и выходных параметров;

2) функция-решатель (SOLVER) – точнее набор таких функций, подходящих для решения различных задач. Эта функция, выполняя один из алгоритмов численного интегрирования, на каждом шаге расчета один или несколько раз вызывает ODEFUN;

3) программа пользователя, подготавливающая необходимые данные и вызывающая функцию-решатель с передачей ей ссылки на ODEFUN. Результаты расчета после необходимой обработки предоставляются пользователю в том или ином виде.

На пути реализации такой концепции в ПЛК возникает ряд проблем, решение которых рассмотрено ниже с иллюстрацией на примере.

1. В рамках традиционного подхода к программированию ПЛК невозможно ответить на вопрос, какой программной единицей (POU, Program Organizational Unit) можно представить ODEFUN, учитывая то обстоятельство, что ссылка на него должна быть передана функции-решателю. Это не может быть, например, экземпляр обычного функционального блока, поскольку все экземпляры «наследуют» один и тот же исполняемый код, а нам нужно, чтобы SOLVER мог решать любую задачу.

Идеальным средством разрешения проблемы стали появившиеся в CODESYS 3.5 *интерфейсы*. Интерфейсы являются расширением стандарта МЭК 61131 и, вероятно, скоро войдут в него. По сути интерфейс – аналог абстрактного класса из языка программирования C++ и предоставляет набор свойств и методов, причем последние не реализованы, а только объявлены. Реализация методов возлагается на функциональные блоки, «осуществляющие» (implement) интерфейс. А поскольку интерфейс может фигурировать в качестве типа передаваемых данных, решатель получает возможность оперировать любым набором дифференциальных уравнений.

В данной работе интерфейс, представляющий ODEFUN, имеет название DynamicObject и состоит из единственного метода RightParts (рис. 3).

DynamicObject.RightParts

```
1  METHOD RightParts: BOOL
2  VAR_INPUT
3      t:LREAL;  (*время*)
4      x:POINTER TO LREAL;  (*указатель на начало массива переменных состояния*)
5      u:POINTER TO LREAL;  (*указатель на начало массива входных переменных*)
6      dx:POINTER TO LREAL;(*указатель на начало массива производных перем.сост.*)
7      y:POINTER TO LREAL;  (*указатель на начало массива выходных переменных*)
8  END_VAR
```

Рис. 3. Объявление метода RightParts интерфейса DynamicObject.

Так как метод является функцией, он должен возвращать некоторое значение. В нашем случае это фиктивное значение типа BOOL, которое никак не используется.

Входная переменная t сообщает ODEFUN информацию о времени. Это необязательно «абсолютное» время (его всегда можно запросить у системы в реализации метода). Возможно, например, это текущее время работы «динамического объекта». Для универсальности переменная имеет тип LREAL. В нашем примере данная переменная не задействуется.

Все остальные входные переменные метода являются указателями, которые, как предполагается, будут настроены на первые элементы соответствующих массивов. В CoDeSys «техника» работы с указателями и массивами практически идентична «технике» языков C/C++, и указатели вполне могут использоваться вместо имен массивов в конструкциях вида x[2] (значение в скобках означает смещение на два размера базового типа указателя в байтах).

Использование указателей накладывает дополнительные расходы на адресацию, но ускоряет передачу параметров при вызове и позволяет разместить переменные в одном месте.

В нашей системе интерфейс DynamicObject будут реализовывать функциональные

блоки Controller (регулятор) и Object (объект управления). Не имея собственного содержания, эти блоки лишь определяют свои «версии» метода RightParts:

```
FUNCTION_BLOCK Controller IMPLEMENTS DynamicObject
METHOD RightParts : BOOL
VAR_INPUT
        t       : LREAL;
        x,u,dx,y        : POINTER TO LREAL;
END_VAR
dx[0]:=u[0];
y[0]:=1.14*dx[0] + 0.908*x[0];

FUNCTION_BLOCK Object IMPLEMENTS DynamicObject
METHOD RightParts : BOOL
VAR_INPUT
        t       : LREAL;
        x,u,dx,y        : POINTER TO LREAL;
END_VAR
dx[0]:=x[1];
dx[1]:=x[2];
dx[2]:=-8*x[0]-12*x[1]-6*x[2]+8*u[0];
y[0]:=x[0];
```

2. В Matlab функция-решатель, будучи вызванной, не прерывает своей работы и не возвращает управления до тех пор, пока не завершит расчет на всем временном интервале. В ПЛК решатель должен работать в реальном времени. Разумеется, ни о каких методах с переменным шагом в таких условиях не может быть и речи. С другой стороны, отсутствует необходимость запоминания результатов расчетов на предыдущих шагах, поскольку теперь этим будет заниматься вызывающая сторона (если это нужно, например, для архивации трендов). Таким образом, решатель в принципе может быть *функцией*, *программой* или *функциональным блоком*. Рассмотрим названные варианты.

Функция не может сохранять никакой информации между вызовами. Элементарные методы численного интегрирования этого и не требуют, но вполне вероятно, что на определенном этапе развития программного обеспечения такая потребность может возникнуть. Можно представить себе, например, алгоритм, приспосабливающийся к поведению динамической системы, для чего ему придется задействовать некоторый объем «долговременной» памяти.

Реализация решателя в виде программы неприемлема по простой причине: программа не может иметь экземпляров, поэтому в проекте может быть только один решатель. В такой системе невозможно эффективно отслеживать два или более процесса, протекающих с разными скоростями (все процессы придется пересчитывать на максимальной скорости единственным решателем).

Функциональные блоки не имеют ограничений, присущих функциям и программам,

поэтому решатель, очевидно, должен быть представлен функциональным блоком. На рис. 4 показан решатель, выполняющий численное интегрирование методом Хойна.

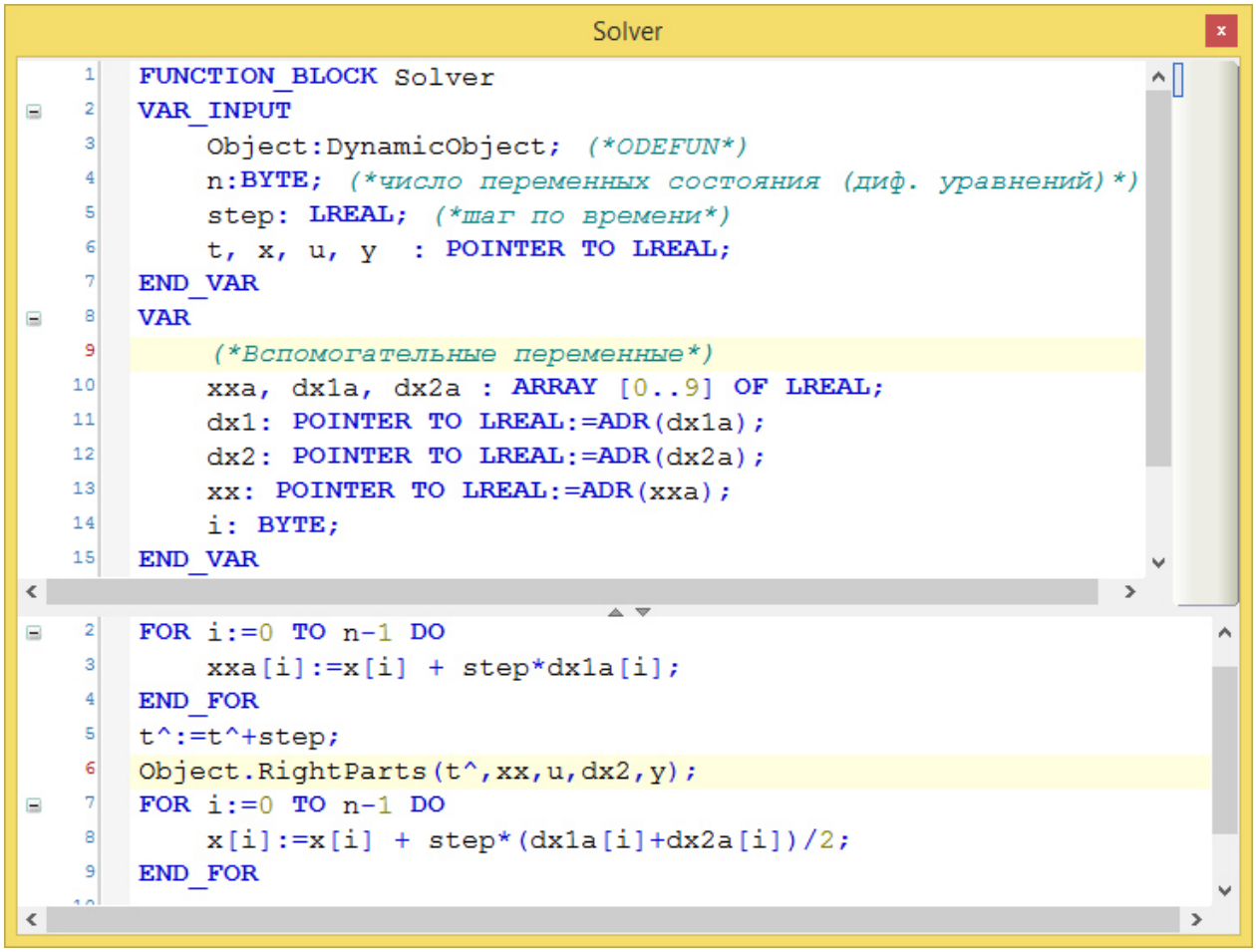

Рис. 4. Функциональный блок – решатель ODE по методу Хойна.

Решатель два раза вызывает метод RightParts интерфейса DynamicObject, переданного ему в качестве входного параметра. Это настоящие *полиморфные* вызовы, распознаваемые не на этапе компиляции, а на этапе выполнения программы.

В списке вспомогательных переменных функционального блока присутствуют три массива из десяти элементов для хранения промежуточных результатов расчетов, необходимых согласно методу. Таким образом, максимальное число уравнений, решаемых решателем, ограничено десятью. Однако поскольку код блока открыт, это число может быть скорректировано как в большую, так и в меньшую сторону в зависимости от потребностей приложения.

3. Пользовательская программа, в которой объявляется экземпляр решателя и из которой он вызывается на исполнение, сама, в свою очередь, должна вызываться циклически. Время цикла должно совпадать со значением входного параметра step функционального блока Solver. Помимо решателя, в программе должны быть объявлены все необходимые экземпляры ODEFUN-блоков, реализующих интерфейс DynamicObject (они будут передаваться решателю при вызовах). Необходимо также подготовить массивы для хранения переменных состояния, входных и выходных переменных динамических объектов. Поскольку здесь речь

идет уже о решении конкретной задачи, размеры массивов известны. В принципе, все объекты могут разделять одни и те массивы, пользуясь разными их частями, что несложно организовать, соответствующим образом настраивая указатели при вызове решателя. Именно такой подход применен при решении иллюстративной задачи (рис. 5).

Результаты моделирования

В тестовой программе, показанной на рис. 5, одновременно функционируют две реализации одной и той же системы. Первая построена на базе традиционного подхода с привлечением библиотечных средств и последовательной обработке сигналов, вторая – на основе технологии численного интегрирования с параллельной обработкой сигналов.

Программа опробовалась на виртуальном ПЛК CODESYS Control Win V3. Для представления результатов разработано окно визуализации, позволяющее в оперативном порядке вводить и плавно изменять задание, наблюдать на графике изменение выходных и управляющих сигналов – как в реальном времени, так и в архиве (рис. 6).

PLC_PRG

```
PROGRAM PLC_PRG
VAR
    sp   : LREAL:=0; (*Задание*)
    (*Решение традиционным способом*)
    yy   : REAL:=0; (*выход подпрограммы "Объект"*)
    uu   : REAL:=0; (*выход подпрограммы "Регулятор"*)
    (*Решение методом численного интегрирования*)
    t    : LREAL; (*время*)
    x    : ARRAY [0..3] OF LREAL := [4(0)];(*x[0]...x[2] - объект, x[3]- рег-р*)
    u    : ARRAY [0..1] OF LREAL := [2(0)];(*u[0] - объект, u[1] - регулятор*)
    y    : ARRAY [0..1] OF LREAL := [2(0)];(*y[0] - объект, y[1] - регулятор*)
    ob   : Object; (*Объект ("ODEFUN")*)
    reg  : Controller; (*Регулятор ("ODEFUN")*)
    sol  : Solver; (*Решатель*)
END_VAR
```

```
(*Решение традиционным способом*)
ControllerCFC(u:=sp-yy,y=>uu);
ObjectCFC(u:=uu,y=>yy);
(*Решение методом численного интегрирования*)
sol (Object:=ob, n:=3,step:=0.02,t:=ADR(t),x:=ADR(x),   u:=ADR(u),   y:=ADR(y));
sol (Object:=reg,n:=1,step:=0.02,t:=ADR(t),x:=ADR(x[3]),u:=ADR(u[1]),y:=ADR(y[1]));
u[0]:=y[1];
u[1]:=sp - y[0];
```

Рис. 5. Головная программа PLC_PRG – решение иллюстративного примера.

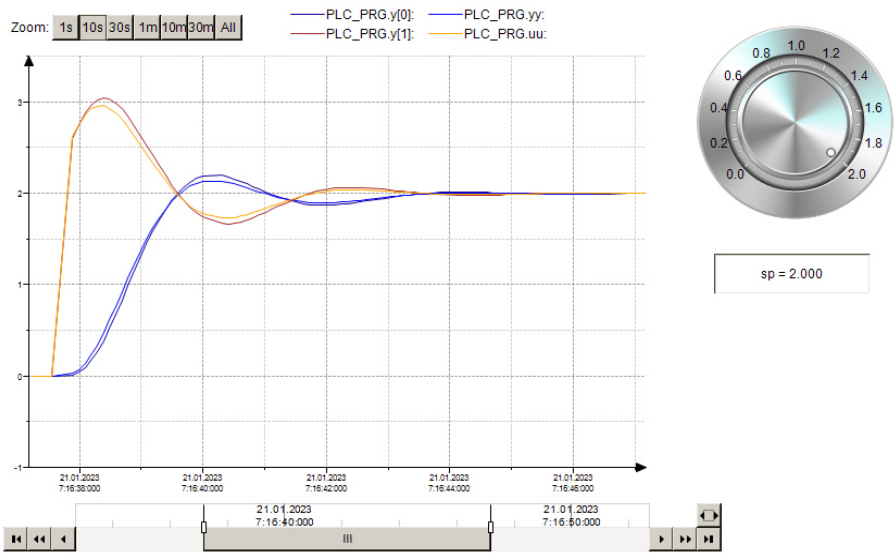

Рис. 6. Экран визуализации – сравнение результатов расчета.

На графике (элемент «Тренд») зафиксирован процесс отработки ступенчатого изменения задания от нуля до двух единиц. Как и следовало ожидать, обе реализации демонстрируют схожее поведение, однако различия графиков достаточно заметны, что говорит в пользу перехода на технологию численного интегрирования как более совершенную и математически обоснованную по сравнению с традиционной.

В примере рассмотрена простейшая линейная система четвертого порядка. Следует ожидать, что по мере увеличения сложности и порядка погрешность, вносимая примитивными алгоритмами интегрирования и последовательным пересчетом блоков, будет только возрастать, тогда как альтернативный вариант обещает увеличение точности за счет применения более совершенных алгоритмов численного интегрирования.

Заключение

Адекватная реализация современных алгоритмов управления на программируемых логических контроллерах и других управляющих машинах требует повышения качества «воспроизведения» цифровых динамических моделей. Применение методов численного интегрирования обеспечивает необходимую точность расчета, а внедрение элементов объектно-ориентированного программирования придает программным конструкциям универсальный характер. В целом предлагаемый подход позволит ускорить и упростить перенос решений, найденных в математических пакетах и средах имитационного моделирования, на целевую платформу для апробации и отладки.

---

1 Рыбалёв, А.Н. Реализации алгоритма самоорганизации систем управления по методу большого коэффициента в комбинированной и контроллерных моделях // Информатика и системы управления. – Благовещенск: АмГУ. – 2023. – Вып. 1(75). – С. 37-49. – Режим доступа: https://ics.pnu.edu.ru/media/2023/N75_04.pdf

2. Кузьмина, Н. Объектно-ориентированное программирование в стандарте МЭК 61131-3 // Современные технологии автоматизации. – 2017. – №4. – С.104-110.

3. Пименов, В.Г. Численные методы: в 2 ч. – Ч. 2 [учеб. пособие] / В.Г. Пименов, А.Б. Ложников; [науч. ред. Ю.А. Меленцова]; М-во образования и науки РФ, Урал. федер. ун-т. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 106 с.