****

**СОЗДАНИЕ СИСТЕМЫ УПРАВЛЕНИЯ**

**НА БАЗЕ НЕЧЁТКОЙ ЛОГИКИ**

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу «Управление в технических системах»

# ВВЕДЕНИЕ

Среда динамического моделирования технических систем SimInTech является инструментом для создания математических моделей любых систем, описание которых может быть представлено в виде систем алгебраических и дифференциальных уравнений в общих производных. SimInTech содержит в себе развитые инструменты, обеспечивающие расширение возможностей по моделированию процессов, в данной работе рассматривается пример создания собственной библиотеки блоков.

В данной работе рассматривается процесс создания алгоритмов управления на базе «нечеткой» логики в среде динамического моделирования технических систем «SimInTech». Показаны методы создания собственных блоков на базе встроенного в SimInTech языка программирования. Демонстрируется возможность генерации кода Си управляющих программ.

# ЦЕЛЬ РАБОТЫ

* демонстрация процесса создания собственных блоков для SimInTech встроенными инструментами.
* ознакомление с новыми методами формирования математических моделей динамики САР в программном комплексе "SimInTech", включая:
  + процедуры создания новых блоков с помощью языка программирования;
  + создание собственных библиотек блоков;
  + создание технической анимации на изображении блоков;
  + генерация кода Си;
* ознакомление с основами создания управляющих блоков на базе нечеткой логики, включая:
  + фазификацию переменных;
  + формирование математической модели базы правил нечеткого вывода;
  + аккумуляцию, агрегацию и нечеткий вывод.

# 1 ОСНОВЫ ТЕОРИИ НЕЧЁТКОЙ ЛОГИКИ

## 1.1 «Нечёткая» логика – это просто

В классической теории автоматического управления, управляющее воздействие на систему рассчитывается в зависимости от регулируемой величины, выраженной в численном виде. Система управления производит преобразование входных параметров в управляющие воздействия передаточными функциями (рассчитываемым по алгоритмам в цифровых системах, либо на основе физических приводов в аналоговых системах управления).

«Нечёткая» логика – это принцип построения алгоритмов управления на базе системы логических правил, аналогичных классической логике. При данном подходе управляющее воздействие рассчитывается на основе набора правил типа:

**Если** *(входной параметр),* **то** *(управляющее воздействие)*;

**Если** *(входная параметр 1)* **и** (*входной параметр 2),* **то** *(управляющее воздействие)*;

Например, для регулирования температуры воды в смесителе эти правила могут выглядеть так:

**1. Eсли** *вода горячая,* **то** *открываем кран холодной воды***;**

**2. Eсли** *вода холодная,* **то** *открываем кран горячей воды***;**

**3. Eсли** *вода холодная* **и** *кран горячей открыт полностью***, то** *закрываем кран холодной воды***;**

**4. Eсли** *вода горячая* **и** *кран холодной воды открыт полностью***, то** *закрываем кран горячей воды***.**

Как видно из примера, управляющее воздействие определяется не в виде вычисления в зависимости от температуры воды, а в виде логического анализа утверждений.

Основная проблема построения алгоритмов на базе нечеткой логики – перейти от параметров системы к логическим утверждениям, как для входных величин (перейти от измеренной температуры к утверждению *вода горячая*), так и для регулирующего воздействия (перейти от *открываем кран* к углу поворота, или скорости вращения вентиля).

Именно процессы этого перехода мы разберем в данной лабораторной работе.

## 1.2 Принцип построения алгоритма нечеткого вывода

Для того, чтобы можно было применять простые правила, данные, передаваемые в блок регулирования на основе нечеткой логики, должны быть преобразованы. Для этого предлагается следующие:

Входные параметры и выходные воздействия преобразовываются в **лингвистические переменные**. Каждая лингвистическая переменная характеризуется набором **термов**. Например, лингвистическая переменная **температура** может иметь следующие термы "*холодно*", "*нормально*", "*горячо*". Лингвистическая переменная уровень может иметь следующие термы: "*низкий*", "*нормальный*", "*высокий*". Лингвистическая переменная **команды задвижки** может иметь следующие термы "*открывать быстро*", "*открывать медленно*", "*не трогать*", "*закрывать медленно*", "*закрывать быстро*". Каждый терм описывается своей **функцией принадлежности** *µi(x),* которая может принимать значения от 0 до 1. Получив значение входной переменной x в блоке нечеткой логики вычисляется значения *µi(x)* каждого терма. Эта процедура называется термином **Фазификация**.

Результатом применения правила является величина, называемая степенью истинности, а попросту число от 0 до 1. Чтобы было понятно, рассмотрим пример: допустим, есть задвижка, которая открывается по сигналу уровня, и простое правило:

Если **уровень** *низкий,* то **команда задвижки** - *открывать быстро*.

У нас есть входная величина – уровень **h** в метрах. Если величина уровня низкая (*µlow*(h) = 1, где µlow(h) - *функция принадлежности* для **терма** *низкий*), или уровень не низкий ((*µlow*(h) = 0), то все происходит как и в обычной логике, степень истинности правила принимает значение 1 или 0. Нечеткость начинается если 0 < *µlow*(h) < 1, например = 0.5. Уровень низкий, но не очень. Соответственно и открывать задвижку нужно быстро, но не очень. В данном правиле *степень истинности* равна 0.5. Подобным образом происходит ее вычисление для каждого правила.

Этот процесс называется **Активизация**.

Заключения из каждого правила собираются вместе для каждой лингвистической переменной, этот процесс называется **Аккумуляция**. Например, после расчета набора правил, мы получаем результаты для лингвистической переменной **задвижка:** "*открывать быстро*" - 0.5, "*открывать медленно*" - 0.3, "*не трогать*" - 0, "*закрывать медленно*" - 0, "*закрывать быстро*" - 0. Понятно, что скорость задвижки лежит где-то между медленно и быстро.

Зная степень истинности для каждого терма выходной переменной можно рассчитать ее числовое значение. Эта процедура называется **Дефазификацией**.

Таким образом, работа блока управления на базе нечеткой логики может быть разбита на следующие этапы:

1. Фазификация входных переменных.
2. Активизация заключений правил нечеткой логики.
3. Аккумуляция заключений для каждой лингвистической переменной.
4. Дефазификация выходных переменных.

Все эти этапы мы и выполним в SimInTech с помощью встроенного языка программирования.

# 2 СОЗДАНИЕ РЕГУЛЯТОРА НА БАЗЕ НЕЧЕТКОЙ ЛОГИКИ В SIMINTECH

## 2.1 Встроенный язык программирования

В 2004 году в SimInTech появился новый блок - *язык программирования*. Данный блок позволяет создавать собственные модели в виде программы, выполняемой на каждом шаге расчета. Таким образом, можно формировать любые модели, описание которых существует виде системы алгебраических и дифференциальных уравнений, без компиляции, непосредственно в среде SimInTech. В языке программирования поддерживаются все основные конструкции языков высокого уровня, а также содержится большой набор математических функций, который постоянно расширяется.

Внешний вид SimInTech с открытым окном редактора языка программирования представлен на рис. 2.1. Новый блок по умолчанию находится в закладке **Динамические**. Редактор языка программирования открывается по двойному клику на блоке в схемном окне.

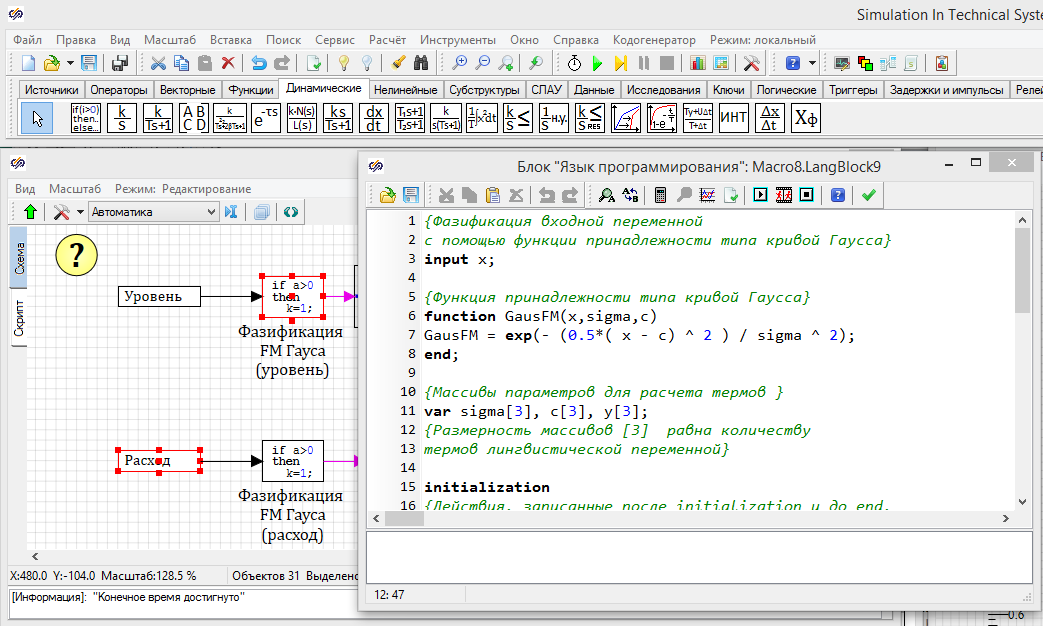


Рис. 2.1. Вид окна языка программирования

## 2.2 Фазификация – простой пример

В качестве первого примера выполним фазификацию входной переменной. Наш блок, на вход будет получать 1 значение и оценивать его, насколько данная величина, соответствует заданным в параметрах термам (например, *большая*, *нормальная* или *малая).* Мы будем расчленять исходную величину на несколько термов, рассчитывая для каждого терма *функцию принадлежности*.

Создайте новую схему автоматики и настройте параметры расчета. Установите время моделирования 20 секунд. Для этого нужно залезть в меню ***Расчет/Параметры расчета*** и найти строку ***Конечное время расчета***. Для расчета с максимальной скоростью, на закладке «*Синхронизация*» снимите галочку «*Синхронизировать с реальным временем*» см. рис. 2.2

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

Рис. 2.2. Настройка параметров расчета

Поместите на схему три блока: Из закладки **Источники** возьмите блок ***Линейный***, из закладки Данные возьмем блок *Временной график*, ну и наконец, самый интересный для нас блок из линейки **Динамические** – блок *Язык программирования*.

Наша задача выполнить фазификацию переменной. Роль входной переменной будет играть линейный источник.

Переменная у нас будет меняться от -1 до 1. Для этого установите параметры блока *Линейный источник:*

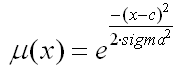
*Свободный член* -1;

*Коэффициент при t* - 0.1.

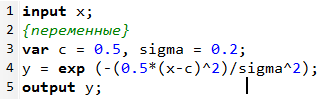
Откройте редактор блока языка программирования двойным кликом на блоке в схеме. Снесите весь текст, который там есть по умолчанию. И напишете простейшую программу - текст как показано на картинке 2.3

|  |  |
| --- | --- |
|  |  |
| Рис. 2.3. Программа блока | Рис. 2.4. Окно SimInTech после расчета модели |

Закройте окно, нажав на кнопку с зеленной галочкой справа вверху. После закрытия видим, что у блока появились *входной* и *выходной* порты. На вход подайте сигнал из блока линейный. Ну а для того, чтобы было, куда подключить выход блока, в свойствах графика увеличьте количество *входных портов* до 2 и соедините выход блока программирования с графиком. Запустите расчет и на графике должен быть косой крест как на рисунке 2.4 Входная переменная возрастает - выходная переменная убывает.

Ну вот в принципе вы сделали свой первый блок-программу, которая инвертирует вход. Теперь сделаем полезную вещь, а именно, расчет *функции принадлежности* ***µ(x)******типа кривой Гаусса***. Формула для расчета известна: . Чтобы запрограммировать эту формулу, необходимо ввести в программу две переменные: с и sigma, и записать код для расчета кривой.

Измените текст в блоке язык программирования на следующий:



Для увеличения наглядности используйте блок *фазовый портрет* из закладки **Данные**. К верхнему входу *фазового портрета* подключите выход линейного блока, а к нижнему – выходы *языка программирования*. См. рисунок 2.5. Запустите расчет, на графиках появилась «шишка», см. рис. 2.6.

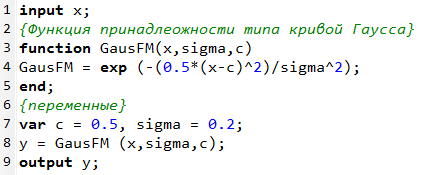
Две переменный ***с*** и ***sigma*** определяют вид кривой: ***с*** - это значение, при котором *функция принадлежности* равна единице - верхушка шишки. ***Sigma*** – параметр ширины шишки.

Таким образом мы получили блок, который позволяет рассчитывает *функцию принадлежности* для входной величины. Из графиков видно, что в момент времени, когда входная переменная х = 0.5 (наклонная кривая) функция принадлежности равна единице, и чем больше х отличается от 0.5 тем меньше *функция принадлежности*. Имея такой блок уже сейчас можно скопировать его на схеме получать разные функции принадлежности для разных термов задавая разные значения ***с*** и ***sigma***.

|  |  |
| --- | --- |
|  |  |
| Рис. 2.5. Схема для расчета функции принадлежности | Рис. 2.6. Графики расчета функции принадлежности |

Для сокращения количества блоков на схемы, мы сделаем из одного сигнала три терма с тремя функциями принадлежности в одном блоке.

Для этого запишем функцию принадлежности как функцию. Измените текст на следующий:



Запустите на счет и убедитесь, что ничего не изменилось. А теперь воспользуемся тем счастливым обстоятельством, что в SimInTech почти всё, что может быть векторизовано, является вектором. Следовательно, и переменные могут быть векторами. Пусть, в базе правил нечеткой логике используем три **терма**: *низкий, нормальный* и *высокий*. Соответственно вместо одного значения на выходе нужно три, каждый из которых показывает насколько величина является, **низкой,** **нормальной** или **высокой**. Делается это легко и не принужденно.

Переопределим переменные **с,** **sigma** и выход **у** в массивы из трех элементов соответственно **c[3], sigma[3], у[3]**. Запишем для каждого трема свои параметры функции принадлежности. И для сокращения вычислений сделаем присвоение значений переменным в блоке инициализации (между слов **initialization** и **end**). Это блок выполняется только одни раз при запуске расчета, незачем повторять присвоение на каждом шаге интегрирования. А теперь в цикле ровно три раза, вызовем функцию принадлежности. Текст всего этого приведен ниже:

|  |  |
| --- | --- |
|  | Если сейчас нажать на кнопку старт, то SimInTech покажет вам ошибку и сообщит что не может привести в соответствие какие-то размерности. Дело в том, что фазовый портрет должен иметь на обоих портах одинаковую размерность, а у нас переменная, попавшая в *язык программирования* расплодилась. Необходимо, размножить переменную подаваемую на верхний вход фазового портрета. Возьмите в закладке ***Векторные*** блок *размножитель*, и поместите его на схему, как показано на рисунке 2.., а в качестве параметров запишите 3#1. Теперь переменная преобразуется в вектор из трех одинаковых значений и фазовый портрет работает (см. рис. 2.8).  Вот и все, с помощью каких то 15 строчек кода мы выполнили фазификацию переменной на три терма. Сохраните эту модель, в дальнейшем можно использовать эту схемы для проверки работы кода, для фазификации переменной другими функциями принадлежности. Результаты моделирования представлены на рис. 2.8. |

|  |  |
| --- | --- |
|  |  |
| Рис. 2.7. Схема демонстрации фазификации | Рис. 2.8. Результаты работы блока фазификации |

Посмотри еще раз на дело рук своих. Мы создали блок, который обеспечивает анализ входной переменной и ее разложение на 3 терма (низкий, нормальны, высокий) (фазификацию). В данном примере принято, что -1 это *низкий* (low), 0, это *нормальный* (normal), 1 это *высокий* (high). В нашей тестовой схеме переменная меняется от – 1 до 1. Запустив расчет, мы наглядно видим, как линейные изменения одной переменной от – 1 до 1 приводят к изменению значений трех термов, в диапазоне от 0, до 1. Вначале, когда входная величина равна -1, терм *низкий* имеет функцию принадлежности = 1, термы *нормальный* и *высокий* равны нулю. По мере роста входной величины, значение терма *низкий* уменьшается (величина еще низкая, но уже не так), а значение терма *нормальный* начинает расти, чем ближе к 0, тем ближе значение терма *нормальный* к *1*. Так на наших глазах, простой рост входной величины от -1 до 1, не имеющий никакой наглядности, превратился в наглядный жизненный процесс перехода от *низкий*, через *нормальный* к *высокий.* А с этими значениями уже можно формировать правила логического вывода.

## 2.3. Создание собственной библиотеки блоков.

В среде SimInTech вы можете создавать свои библиотеки, которые могут продаваться как решение третьих сторон. Покажем, как это делается на примере библиотеки нечеткой логики. На предыдущем этапе мы создали блок, который осуществляет фазификацию входной переменной на три терма, понятно, что меняя константы в коде, мы можем фазифицировать переменную на любое количество термов.

Сейчас создадим блок, в котором пользователь будет задавать сколько нужно ему термов и какие значения им соответствуют. Что бы не изменить, то что уже работает, в SimInTech первым делом необходимо сменить название класса блока. Войдите в Свойства блока «*Язык программирования»,* который мы с вами сделали ранее, и замените тип элемента дав ему название «**Фазификация Гаусса**» (см. Рис. 2.9) Теперь при сохранении этого блока в библиотеку, мы не изменим существующий блок «*Язык программирования*».

|  |  |
| --- | --- |
|  |  |
| Рис. 2.9. Именуем новый блок | Рис. 2.10. Переводим систему в режим разработчика |

После того как тип элемента переименован, можно приступить его редактированию. Для начала нужно перевести SimInTech в режим разработчика. Заходим в главное меню программы пункт «*Файл*» подпункт «*Параметры*» В появившемся окне нужно поставить галочку в опции «*Режим разработчика*» (Рис. 2.30).

В режиме разработчика выделяем блок «*Язык программирования*» на схеме и переходим к главному меню программы «*Правка*» в самом низу находится подпункт «Изменить блок», в диалоговом окне необходимо снести все свойства, кроме свойства «*Транслировать скрипт в код автоматически*». (Данное свойство обеспечит нам возможность получить Си код из блока.) После этого добавить новые как показано на Рис. 2. 11. Для удаления и добавления используются кнопки внизу справа. В качестве свойств блока мы задаем **Количество термов**, **Mассив значений с**, и **Массив значений sigma.** Тут же мы даем значения по умолчанию (см. Рис. 2.11).

|  |  |
| --- | --- |
|  |  |
| Рис 2.11. Свойства блока фазификация, заданные разработчиком блока | Рис 2.12. Свойства блока фазификация |

Нажимаем кнопку **Ок**. Если вы все сделали правильно, у нашего блока в свойствах появились ровно те же поля. Что бы посмотреть и поменять свойства, выделите блок, кликните правой кнопкой мыши, в всплывающем меню и выберите пункт «*Свойства объекта*». В окне выберите закладку «*Свойства*» (см. Рис 2.12). И вот оно счастье, меняйте параметры и получайте разнообразные фазификации в любом количестве.

Перейдем в окно редактирования скрипта и изменим текст нашей программы с учетом, того что у нас появились новые переменные, которые будут задавать счастливые пользователи нашего блока.

Вместо задания переменных в тексте программы сделаем присвоении их из переменных, заданных пользователем. Добавляем защиту от отрицательного ввода кол-ва термов. Защита от дурака не когда не мешала, хотя нельзя предусмотреть изобретательность дурака. Сделаем еще одну защиту от выхода входной величины за границу диапазона.

Поясню, если входная величина равна границе диапазона, то согласно алгоритму, значение функции принадлежности для крайних термов равно единице. Однако, при выходе за границу диапазона значение функции принадлежности начнет уменьшатся, шишка Гаусса идет на убыль. Очень часто, это является ошибкой. Например, входная температура разбита на три терма: **холодно** +10 C, **нормально** +30 C, и **горячо** +55 C. Если температура 55 С это на 100% **горячо** (функция принадлежности = 1), и по правилам логики нужно максимально быстро крутить кран холодной воды, то при температуре 100 C, явно нужно делать тоже самое, хотя функция гаусса в этой точке может превратится в ноль. Чтобы этого не случилось введена проверка выхода значения входной величины за границу диапазона. Текст всего это лежит приведен ниже.

|  |  |
| --- | --- |
|  | Теперь, если сохранить этот блок и запустить на расчет модель, графики должны быть точно такими как на рис. 2.8. Если это так, то поздравлю вы только что разработали новый блок для SimInTech и сделали первый шаг к тому что бы стать миллионером. Что бы убедиться в этом чуде, поменяйте свойства блока на другие и посмотрите какие графики у вас рисуются. Например, для 5 термов, при параметрах, заданных для блока, как на рисунке 2.13, у вас и графики похожи, то вы все сделали правильно.  Теперь можно заносить этот блок в библиотеку, но перед этим изменим его свойства, что бы он был более похож на стандартные блоки. Выделите блок на схеме, правой кнопкой мыши вызовите всплывающее меню и выберите пункт «*Свойства объекта*». В окне выберите закладку «*Общие*». В нижней части списка свойств можно изменить «*Графическое изображение*» блока. Рекомендую так же установить **Да** для свойства «*Блокировка доступа*». В этом случае, при двойном клике по блоку на схеме будет вызываться окно свойств, а не текст программы (см. рис. 2.14), который уже настолько хорош, что не требует правки со стороны пользователя блока. |

|  |  |
| --- | --- |
|  |  |
| Рис. 2.13. Фазификация на 5 термов функция гаусса | Рис. 2.14 Настройка блока фазификация перед добавлением в библиотеку |

Для занесения блока в библиотеку, выделяем его на схеме и в главном окне выбираем пункт меню «*Файл*» подпункт «*Сохранить в библиотеку*».

(Я бы еще рекомендовал «*Сохранить в файл*», чтобы потом продавать это файл). Блок сохранился в библиотеку, но мы пока его не видим в линейке. Еще раз меню «*Файл*», потом «*Редактировать библиотеку*». Пролистываем столбец «*Все записи*» и в самом низу находим наш блок «*Фазификация Гаусса*» (см. Рис. 2.15):

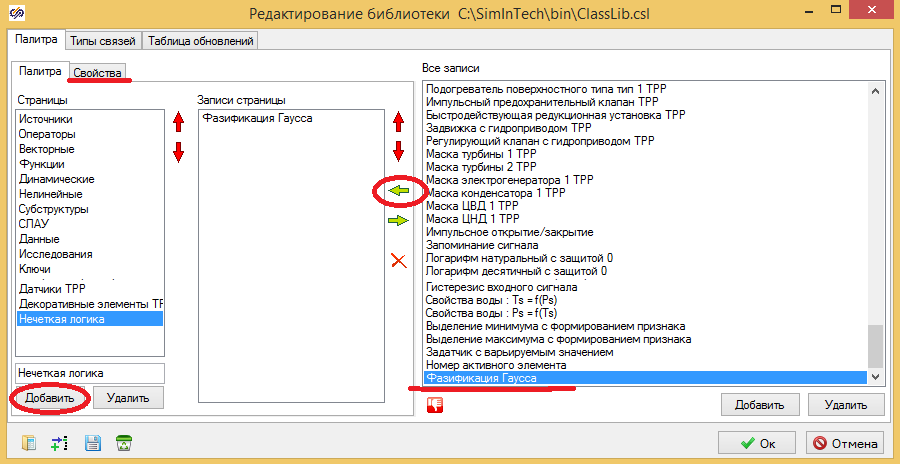


Рис. 2.15 Создание новой закладки “Нечетка Логика”

Ну вот последние клики мышкой добавляем страницу палитры “*Нечеткая Логика*” переносим в нее наш блок, на закладке «*Свойства*», задаем ему картинку и вуаля, у вас в палитре новая библиотека. (См. Рис. 2.16):

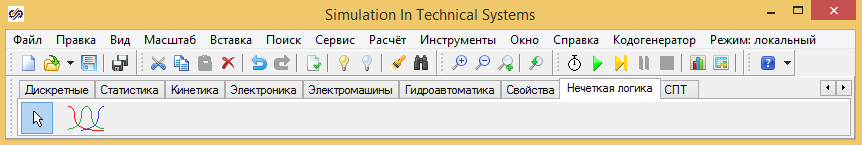
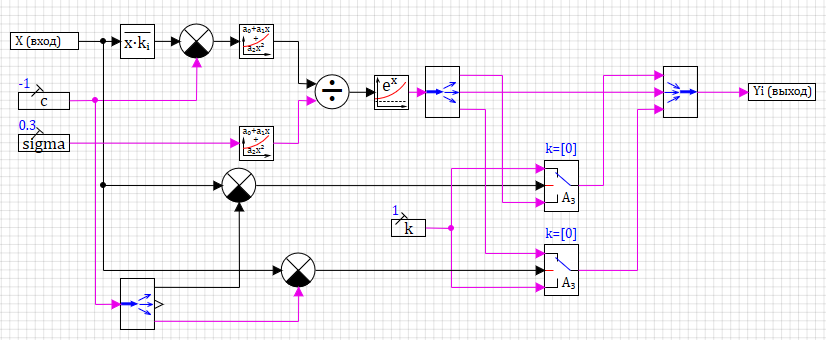


Рис. 2.16. Новая линейка блоков.

SimInTech позволяет создавать совой блоки и в виде схемы, в данной работе мы рассматривали создание блока на базе «*Языка программирования*», но тоже самое можно было сделать на базе блока «*Субмодель*» используя стандартную библиотеку блоков в, например так выглядит схема реализующая блок «*Фазификация Гаусса*» в виде схемы:



## 2.4. Дырявый бак, управляемый нечеткой логикой

Прежде, чем перейти к овладению следующим этапам работы алгоритма нечеткой логики, сделаем объект управления. Управлять будем уровнем в дырявом баке. Задача проста как в школе про бассейн, в одну трубу вливается в другую выливается. Мы будем управлять расходом входа, а утечка будет происходить самотеком.

Параметры бака: высота - 2 м; Площадь сечения - 1 м; Диаметр выходного отверстия 0.05 м2.

Создайте новый проект, поместите на схему «*Субмодель*» из закладки «*Субструктуры*». Войдите в блок, переключитесь на закладку «Скрипт» схемного окна и введите данные по баку:

|  |  |
| --- | --- |
|  | Динамику бака можно описать следующими уравнениями: |

где: **V** - объем бака м3; **rate** - расход поступающий в бак м3/с; **outrate** - расход утечки м3/с; **h** - уровень в баке м; **area** - площадь сечения бака; **outarea** - площадь выходного отверстия; **g = 9.81** - ускорение свободного падения.

Бросьте на схему наш любимый блок «Язык программирования» и запишите уравнения динамики:

|  |  |
| --- | --- |
|  | Вот, чем особенно хорош блок Язык программирования SimInTech, теперь пользователь как узбек (что вижу, о том пою) что видит, то и моделирует. Если уравнения динамики известны, их нужно просто записать, не забывая указывать начальные условия для динамических ременных. Обратите внимание что уравнение для производной записываться так же как и обычные уравнения в языке программирования, отличие только в знаке ‘ после имени переменной.(строка **V’ = rate - outrate**). Так же для дифференциальной переменной необходимо задать начальные условия (здесь **init V=0**) и все, после этого динамика модели ядром SimInTech. |

Соберите схему, вокруг модели, как показано на рисунке и модель бака дырявого бак готова.

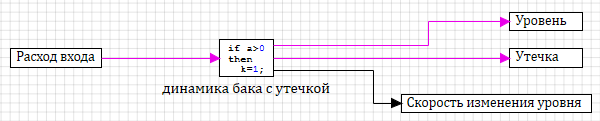


Рис. 2.17. Cхема субмодели дырявого бака

Разберемся теперь с клапаном управляющим расходом. Основная идея такова. Задается номинальный̆ расход 0.5 м3/с. Управляющие воздействие это скорость закрытия-открытия задвижки, интегрируя данную скорость мы получаем положение задвижки - число от 0 (задвижка закрыта - расход равен 0) до 1 расход равен номинальному. Зависимость расхода от положение задвижки линейная. Бросьте на основную схему новый *блок «Субмодель»* из закладки **Субструктуры** войдите в него и соберите схему как на рис. 2.18:

|  |  |
| --- | --- |
|  |  |
| Рис. 2.18. Cхема субмодели клапана | Рис. 2.19. Общая схема модели |

Теперь модель объекта управления типа дырявый бак, готова к работе. Осталось положить на схему недостающие блоки, для задания уровня, отображения результатов. Конечная схема модели, вместе с регулятором уровня на базе нечеткой логики будет выглядеть следующим как на рисунке 2.19.

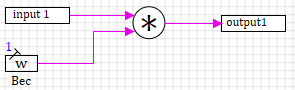
Работа системы проста и незатейлива. Блок «*Меандр*» из закладки «Источники» задает изменение уровня с частотой 20 сек с 1.5 до 0.5 и обратно, заданное значение сравнивается с реальным и разница подается на первый вход блока управления. На второй вход подается скорость изменения уровня в баке. На выходе мы получаем команду клапана который и управляет расходом в бак.

## 2.5. Создание правил регулирования.

Получив модель дырявого бака, самое время перейти к рассмотрению базы правил нечеткой логики. Для нашего бака правила блок управления на базе нечеткой логики имеет входные лингвистические переменные **уровень,** и **изменение уровня.** Переменная **уровень** будет иметь следующие термы: 1 *-высокий*, 2 *-нормальный*, 3 *-низкий*. Переменная изменение уровня будет иметь такие термы: 1 - *уменьшается*, 2 *-не изменяется*, 3 -*увеличивается*. Блок управления клапаном будет выдавать лингвистическую переменную **команда клапана,** которая в свою очередь, будет иметь следующие термы**:** 1 -*закрывать быстро,* 2 -*закрывать медленно*, 3 *-не менять,* 4 *-открывать медленно,* 5 *- открывать быстро.* База правил для управления будет иметь следующий вид:

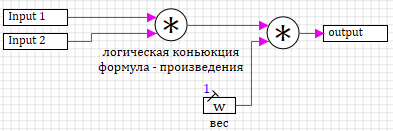
1. **ЕСЛИ** **уровень** *= высокий* **TO** **команда клапана** *= закрывать быстро*;
2. **ЕСЛИ** **уровень** *= нормальный* **TO** **команда клапана =** *не изменять*;
3. **ЕСЛИ** **уровень** *= низкий* **ТО** **команда клапана =** *открывать быстро*;
4. **ЕСЛИ** **уровень** *= нормальный* **И****изменение уровня** = уменьшается **ТО** **команда клапана =** *открывать медленно*;
5. **ЕСЛИ** **уровень** *= нормальный* **И****изменение уровня** = увеличивается **ТО** **команда клапана =** *закрывать медленно*;

Как видим на словах все абсолютно ясно и просто. Поскольку логика у нас нечеткая, для каждого правила нужно, не просто получить ответ да или нет, а рассчитать степь истинности. Для правил 1-3 степень истинности равна величание функции принадлежности для соответствующего терма. Попросту, если уровень h *высокий* на 0.5 (µhigh(h) = 0.5) то и степень истинности *закрывать быстро* равна 0.5. Функциональная схема макроблока реализующего данное правило:



Константа вес позволяет задать весовой коэффициент для правила в общей базе правил. Создайте новую субмодель, и повторите данную схему, после этого данный блок можно положить в библиотеку блоков.

Степень истинности правил 4 и 5 зависит от двух входных переменных, соединенных логическим И, по научному можете называть это логической конъюнкцией. Для вычисления степени истинности существует несколько вариантов, мы используем метод алгебраического произведения смотри схему субмодели:



Эту субмодель тоже можно положить в библиотеку готовых блоков на страницу нечеткая логика. Так вы постепенно приближаетесь к созданию собственной бибилиотеке. Имея всего три блока в библиотеке можно уже начинать собирать систему управления на базе нечеткой логики. Возьмите блоки из нашей новой библиотеки и соберите схему как показано на рисунке 2.20.

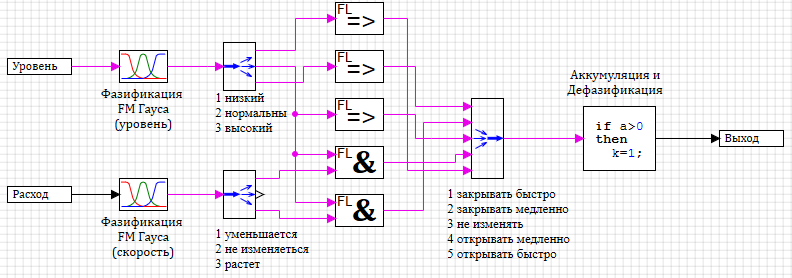


Рис. 2.20. Cхема регулятора на базе нечеткой логики.

На рисунке 2.10 картинки блоков для наглядности изменены, что совсем не влияет на функциональное наполнение блока. Возьмите блоки из библиотеки и перетащите их на схему. Задайте параметры фазификации Гаусса для переменной **уровень**:

*Количество термов* = 3, *Массив значений с* = [-1,0,1], *Массив значений sigma* = 3#0.3 (три раза по 0.3);

для переменной **скорость изменения:**

*Количество термов* = 3, *Массив значений с* = [-0.1,0,0.1], *Массив значений sigma* = 3#0.03 (три раза по 0.03);

Самая большая сложность на этом этапе, это соединение блоков фазификации и блоков правил в нужном порядке. Вот здесь и пригодится возможность подписывать блоки. Выход из блока фазификации для каждой переменной соедините с блоком *Демультиплексор,* для удобства дальнейшей работы с базой правил, подпишите под блоком соответствующие название термов в том порядке в котором они расположены в векторе. Ибо ORDNUNG, ORDNUNG UBER ALLES, как говорят немцы.

Поместите на схему необходимое количество блоков правил.

Выходная переменная имеет пять термов их мы будем предавать в виде вектора. Поместите на схему блок Мультиплексор и задайте 5 входных параметров. Подписи под блоком помогут правильно подключить результаты блоков правил. Помните ORDNUNG, ORDNUNG UBER ALLES!

Соедините схему согласно записанных правил, как показано на рис. 2.20. И сохраните заготовку.

## 2.6. Аккумуляция и дефазификация в одном флаконе

Получив вектор состоящий из пяти заключений из правил нечеткой логики, необходимо рассчитать значение выходной переменной. Для этого мы воспользуемся алгоритмом Мамдани.

Для начала осуществим фазификацию выходной переменной на пять термов с помощью треугольных функций принадлежности. Как Вы уже догадались мы снова будем пользовать блок Язык программирования. Первая часть текста программы приведена на рисунке 2.21

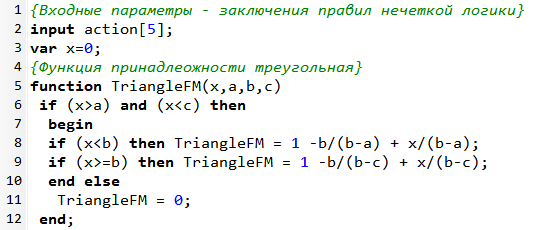


Рис. 2.21. Треугольная функция принадлежности для термов вывода

action[5] - это массив входных значений (заключений из базы правил), который в процессе моделирования мы получим из блока Демультиплексор;

var x=0; объявление переменной для тестирования функции, непосредственно в окне реадактора.

TriangleFM - треугольная функция принадлежности.

Записав данный кусок кода не плохо бы проверить, что мы нагородили, и тут нам на помощь средства проверки зависимостей в языке программирования SimInTech. Для этого мы и ввели переменную x, потом ее можно удалить после завершения тестирования.

За что инженеры-физики уважают MachCad? Не в последнюю очередь за то, он позволяет проверить расчет, непосредственно в процессе записи. В любой момент можно построить графическую зависимость и посмотреть не пропущено-ли где возведение в степень, не стоит ли знак плюс вместо минуса или умножение вместо деления. А вот при программировании постоянно приходится, вооружившись дебаггером, шерстить код на предмет ошибок...

Чтобы убедится в отсутствии багов в написанном куске кода, построим график функции. Для этого нажмите на кнопку  «*построить графическую зависимость*». При этом появится диалоговое окно построения зависимости как на рис. 2.22, Вообще то это окно предназначено для обработки векторных данных, но если поставить галочку в окошке Скалярная зависимость, то "легким движением руки шорты превращаются в..." графопостроитель. В верхней строке задаете минимальное, максимальное значения по оси их X и количество точек графика. В нижнем записываете функцию которую только что запрограммировали. Нажимаете кнопку ОК и видите результат как на рис. 2.22:

|  |  |
| --- | --- |
|  | Теперь меняя значения параметров функции (a, b, c) и нажимая кнопку ОК можно получать графики функции при различных параметрах.  Следующая часть листинга приведена на рисунке 2.23. Здесь описываются параметры необходимые для нечеткого вывода:   * Количество термов выходной величины; * Минимальное и максимальное значение аргумента функции выходной переменной (оно же является диапазоном для численного интегрирования). * Массивы переменных для треугольных функций фазификации выходной переменной. * Относительную точность интегрирования. * Рассчитываем шаг интегрирования. |
| Рис. 2.22. Проверка работы функции графопостроителем. |

Для удобства дальнейшей проверки присваиваем значения входному вектору action. (Дело в том, что до начала расчета значения входных переменных в векторе action[5] равны нулю. В процессе расчета эти значения получаются из входного расчета. Для проверки кода без запуска моделирования можно присвоить им проверочные значения, если это сделать в секции initialization, то в последствии это присвоение можно не удалять так как на расчет они не повлияют.

|  |  |
| --- | --- |
|  | Параметры a[i], b[i], c[i], задаются с учетом расположения заключений правил в входном векторе смотри рис 2.10. В примере принято расположение термов по возрастанию. В принципе, расположение может быть любое. Главное соблюдать соответствие между индексом терма в векторе входа, и индексом параметров функции принадлежности терма. |
| Рис.2.23. Задание параметров для нечеткого вывода |  |

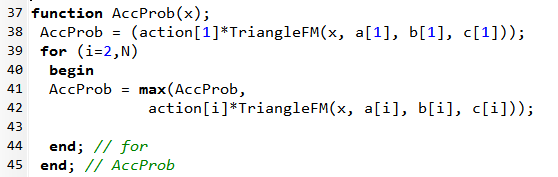
Соответствие термов и параметров их функций принадлежности приведены в таблице:

|  |  |  |
| --- | --- | --- |
| **Index** | **Терм** | **Параметры функции принадлежности** |
| *1* | *закрывать быстро* | (-1, -0.9, -0.8) |
| *2* | *закрывать медленно* | (-0.6, -0.5, -0.4) |
| *3* | *не изменять* | (-0.1, 0, 0.1) |
| *4* | *открывать медленно* | (0.3, 0.4, 0.5) |
| *5* | *открывать быстро* | (0.8, 0.9, 1) |

На рис. 2.23 показано заполнение параметров функций фазификация выходной переменной по термам, согласно таблице. Следующая процедура заключается в применении заключений из правил нечеткой логики. Данная процедура называется **активизация**.

Есть несколько методов активизации мы воспользуемся методом prod-активизации, при этом результирующая функция получается умножением степени истинности заключения из правила на соответствующую функцию принадлежности. Например если **степень истинности** для заключения *закрывать быстро* равна **0** (первое значение в массиве **action** ) то после умножения функция принадлежности превратится в ноль.

Одновременно с **активизацией** мы проведем **аккумуляцию -** найдем общую функцию принадлежности для выходной переменной по всем термам. Мы строим общую функцию для заданного диапазона значений. Для этого, функцию принадлежности каждого терма µi(x) нужно умножить на значение **степени истинности** соответствующего заключения (**активизация**), и найти максимальное значение из всех таких произведений для каждого терма в т. x (**аккумуляция**). В результате мы получаем огибающую функцию. Алгоритм вычисления проведен ниже.



Если сейчас запустить построитель зависимостей и построить зависимость для AccProb, то мы получим график как на рис. 2.24. Функция объединила все треугольники теремов, при этом высота треугольников, стала зависеть от значения вектора action, значения которого мы присвоили в секции инициализации (см. рис. 2.23) во время моделирования эти значения будут поступать из блоков расчета правил нечеткой логики, и в каждый момент времени моделирования высота треугольников будет соответствовать результатам логического агрегирования, соответствующих правил.

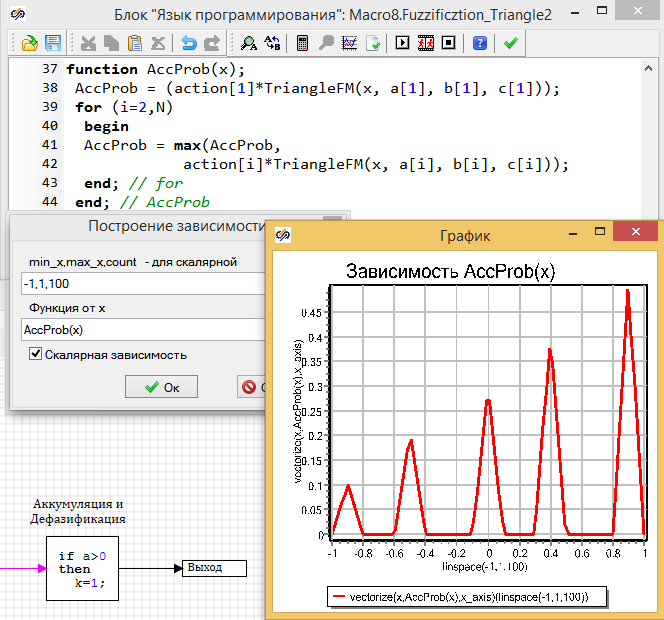
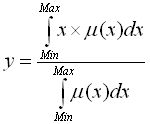


Рис.2.24. Расчет аккумулирующей функции

Таким образом, получив на вход 5 значений вычисления логических правил, и имея треугольные функции принадлежность мы создаем кривую из которой мы должны получит единственно значение выходной величины, для в алгоритме Мамдани, предлагается вычислить центр тяжесть полученной фигуры. Кривая у нас в виде функции AccProb(х) и мы находим центр масс полученной фигуры. Для этого нам и пригодится значение dX, рассчитанное в секции **initialization** блока см.рис. 2.23 Формула расчета центра масс проста:



Ну, а численный алгоритм реализующий эту формулу методом треугольников представлен на рисунке 2.25.

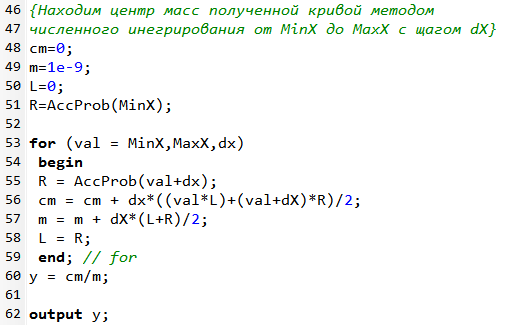


Рис.2.25. Дефазификация методом Мамдани.

Если вы правильно переписали листинги кода с рисунков выше по тексту, то при запуске расчета, на графике, подписанном нами как уровень, из основной схемы, (смотри рис. 2.19.) отразится изменения заданного уровня (синий график) и уровня обеспечиваемого регулятором нечеткой логики (красный график):

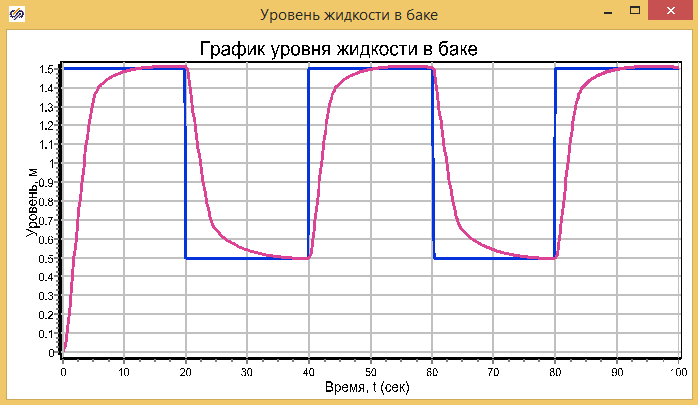


Рис. 2.26 Работа регулятора на базе нечеткой логике для регулирования уровня в дырявом баке.

Поздравляю! Вы только что своими руками создали регулятор на базе нечеткой логики. Для этого Вам пришлось написать около 70 строчек кода. При этом Вы мимоходом, получили три блока которые могут пригодится при решении подобных задач в будущем. После выполнения это лабораторной работы, Вы можете спокойно начинать изучение нечеткой логики, пользуясь любым учебником. Вы теперь знает, что нечеткая логика это не просто, а очень просто особенно с SimInTech. Удачи!

Но это еще не все, поскольку SimInTech это не Simulink, более передовая система мы в следующем разделе покажем как можно еще улучшить наш блок, перед тем как помещать его в библиотеку блоков.

# 3 АНИМАЦИЯ БЛОКА НЕЧЕТКОЙ ЛОГИКИ.

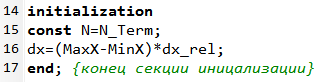
Вот что мне не хватает в этом волшебном блоке, созданным за пару часов? А не хватает наглядности. Например если я ошибся в цифрах, запятую не там поставил. Есть конечно программисты, которые как в матрице, глядя в цифры видят там голую женщину. И это хорошо, но для решения задачи желательно видеть женьщину, а форму кривой которую мы задаем в виде цифр, и желательно до моделирования, и SimInTech это обеспечивает. Сейчас покажу как.

## 3.1 Задание параметров нового блока.

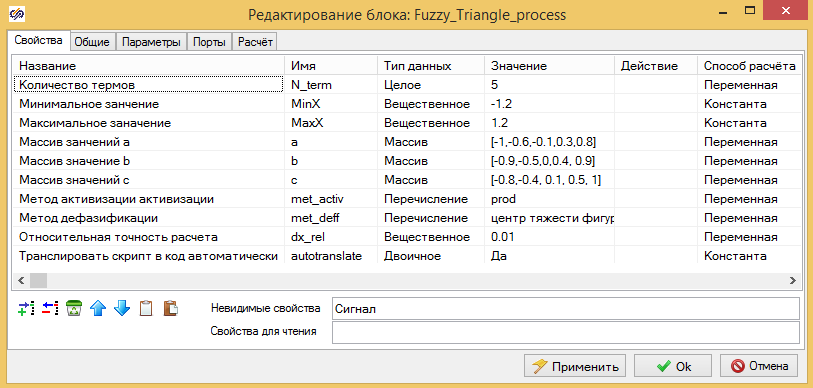
В п. 2.3 описано, как создавать новый блок. Из нашего блока мы создаем новый задав ему имя «Вывод треугольными функциями». Что бы он стал универсальным, вы выносим параметры треугольных функция из текста программы в свойства блока, туда же отправляем в качестве параметров, количество термов, максимальное и минимальное значение выходной величины. Как это сделать см. п.2.3. В итоге свойства блока должны выглядеть так как на рис. 3.1 Значения по умолчанию берем из значений заданных в секции инициализации.

Свойство «*Транслировать в код автоматически*» оставляем из свойств блока «*Язык программирования*» (см п.2.3) нам это свойство пригодиться в дальнейшем.

Как только вы задали эти свойства для блока, становиться абсолютно не нужным объявление этих переменных в тексте программы и можно смело их удалять. Секция инициализации программы сократиться до трех строчек кода:



Для анимации блока нам еще нужна переменная - вектор входных значений, полученных из правил нечеткого вывода, а так же параметр, равный результату нечеткого вывода. Их можно добавить их в закладке параметры окна «Редактирования блока» блока. см. Рис 3.2, параметры отличаются от свойств тем, что их не пользователь задает, а сам блок рассчитывает.



3.1 Свойства блока нечеткого вывода треугольными функциями.

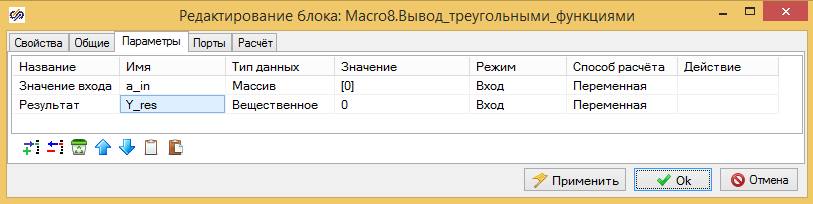
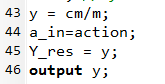


Рис. 3.2 Параметры блока нечеткого вывода треугольными функциями.

В тексте программы блока добавляем присвоение этим переменным:



Таки образом мы подготовили все для начала работы с анимационной системой SimInTech.

В принципе, все что мы будем делать дальше не будет относится к расчету и блок можно закрыть для доступа. Для этого в свойствах блока мы устанавливаем поле «*Блокировка доступа*» - **Да**. После этого двойной клик по блоку будет вызывать окно свойств блока, а не текст скрипта. Потом всегда можно снять блокировку и поправить там, если что то пойдет не так.

Для того что бы оживить наш блок устанавливаем свойство «*Анимированный*» - **Да**. В этом случае во время расчета будет вызваться и исполняться скритп на встроенном языке программирования, обеспечивающий анимацию изображения блока.

Установив эти поля, переходим в режим редактирования графического изображения блока, для этого в поле «*Графическое изображение*» кликаем по кнопке вызова редактора. (см. рис 3.3)

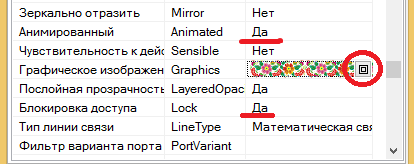


Рис. 3.3. Настройки блока нечеткого вывода

Внешний вид редактора изображен на рис. 3.4. Кроме этого на рисунке присутствует «*Панель примитивов*» из которых можно собрать любое изображение (даже картину), если есть художественный вкус. Тут же на рисунке есть окно редактирования объекта «*Свойства*», в котором можно задать свойства этих примитивов. (см. рис 3.4)

Для вызова панели примитивов используется пункт меню «*Вид*» подпункт «*Панель примитивов*», редактор «*Свойства*» вызывается нажатием правой кнопки мыши, при любом выделенном объекте в «*Графическом редакторе*». На рис. 3.4 нарисована объект-примитив линия и показаны ее свойства.

Для успешного рисования нужно понимать, что окно редактирования содержит в себе, все то пространство, которое будет помещено в рамки блока на схеме. Все что ваше воображение нарисует в этом окне, будет нарисовано и на блоке в соответствующем масштабе. Вы можете настроить размер этого окна и масштаб отображения в нем любым способом как вам удобно.

Открыв это окно сотрите все, что там есть, нам это не пригодится, настройте размеры как вам удобно рисовать и проведите линию по диагонали. Зачем? Все просто, в свойствах это линии, можно прочитать размеры окна в условных единицах измерения. В моем случае свойства линии «*Координаты точек*» показывают что высота данного прямоугольника по оси Y - 32 (от 16 до -16), а ширина по оси Х - 48 (от -24, 24). Таким образом можно догадаться что, центр координат находится в центре, а самое главное я теперь знаю, что для того чтобы мое изображение влезло в прямоугольник блока на схеме, оно должно быть иметь размер не больше (32 Х 48). После этого линию можно удалить.

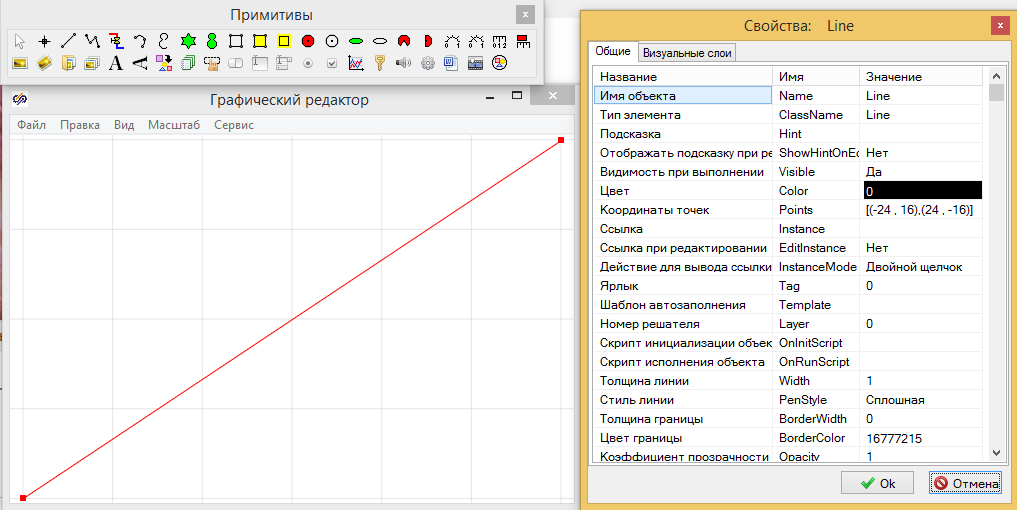


Рис. 3.4. Проверка размеров изображения блока.

Для того, что бы наш блок мог отобразить функции принадлежности, нам необходимо передать в скрипт все параметры, по которым они рассчитываются. Делается это просто. В «*Графическом редакторе*» выберем пункт меню «*Сервис*» далее подпункт «*Cигналы*», в появившемся диалоговом окне мы можем задать сигналы, которые будут использованы скриптом для рисования. Имена сигналов могут быть произвольными, но если задать имена сигналов совпадающих с свойствами блока, то автоматически их значения, во время расчёта будут забираться из свойств блока, что нам и требуется, поэтому просто повторяем, то что у нас есть в свойствах и не забываем параметры, который мы специально сделали и присвоили в скрипте блока для этой цели a\_in и Y\_res. (см. рис. 3.5)

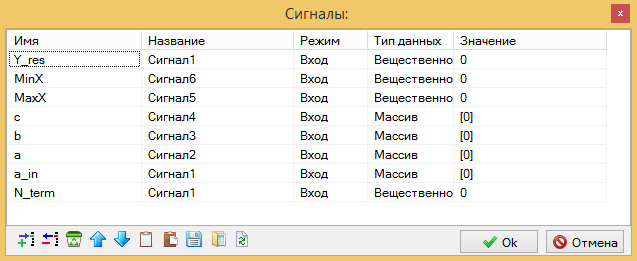


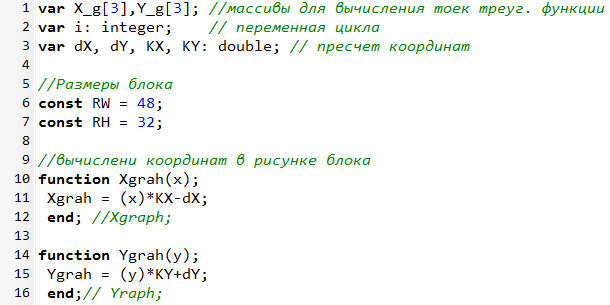
Рис. 3.5. Сигналы передаваемые в скрипт для построения изображения.

## 3.2 Анимация блока, опять скрипт

После того как вы подготовил сигналы для скрипта перейдем непосредственно к программированию анимации. Пункт меню окна «*Графический редактор*», «*Сервис*» подпункт «*Скрипт*» нажимаем и что мы видим? Да, да, это тоже самое окно языка программирования, в котором мы уже прогали математику нечеткой логики. И мы точно так же будем сейчас прогать анимацию, пользуясь тем же самым языком программирования. Все отличие заключается в том, что этот скрипт не рассчитывает выход блока по входу, а управляет окном анимации. И использует сигналы которые мы только что задали.

Для начала подготовим переменные и функции.

Переменные **X\_g[3],** **Y\_g[3]** будут содержать координаты точек, в системе координат графического блока, по которым мы будем строить функции. Переменные **dX, dY, KX, KY**, это коэффициенты пересчета из значений функций в математическом блоке в значения координат на графическом изображении блока. Константы ширины и высоты блока мы выяснили когда проводили линию по диагонали (см. Рис 3.4).



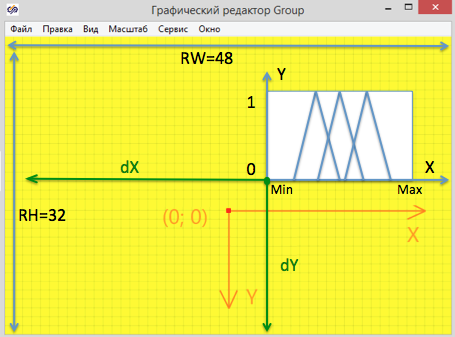
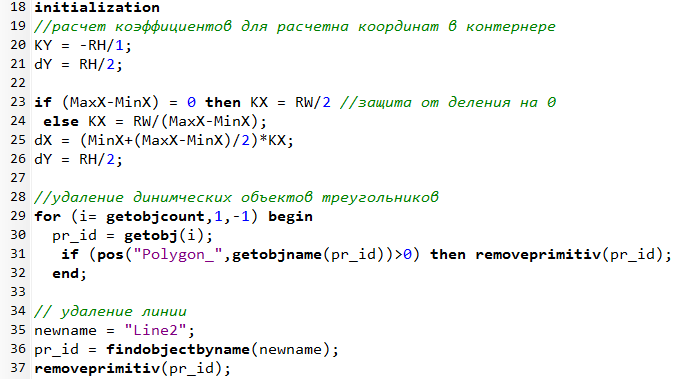


Рис. 3.6 Координатная сетка графического окна блока.

Для того что бы пересчитать величины расчетные в масштаб блока посмотрим на рисунок 3.6. Точка 0,0 находится в центре прямоугольника, направление координат показано красными стрелками (см. Рис. 3.6).Проще всего с координатой Y, поскольку функция принадлежности меняется от 0 до 1, то коэффициент масштабирования KY = - RH/1 (минус, потому что ось Y в графическом окне направлена вниз, см. Рис. 3.6), а для того что сдвинуть О к нижней границе блока DY = RH/2.

Для оси X коэффициент масштабирования KX = RW/(MaxX-MinX), а что бы сдвинуть MinX к левой границе блока нужно от текущего значения отнять DX = (MinX+(MaxX-MinX)/2)\*KX (см. Рис. 3.6) Таким образом значение MinX окажется на левой границы изображения. Что бы не вычислять на каждом шаге коэффициенты мы это сделаем в секции инициализации. В итоге текст будет такой:



Ну и наконец, мы выполнив расчет коэффициентов можно начать рисовать треугольники. И рисовать мы будем не из примитивов и руками, а как настоящие программисты из скрипта и функциями. Задайте следующий текст в секции инициализации:

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

Что делают эти строчки понятно из рисунка треугольной функции. Мы выбрали первую треугольную функцию ***i=1*** и рассчитали координаты трех точек с помощью функций ***Хgrah*** и ***Ygraph,*** а потом создали примитив полигон использую стандартную функцию ***createprimitiv*** передав ему координаты точек в качестве параметров. Закройте окно нажав кнопку «*Применить*», закройте «*Графический редактор*», на вопрос «*Сохранить изображение?*» отвечайте да, и будет вам чудо как на рис. 3.7 (Блок увеличен для наглядности). В поле блока вдруг откуда не возьмись появлялся треугольник построенный по точкам, заданным в параметрах блока и отмасштабированный в размер блока.

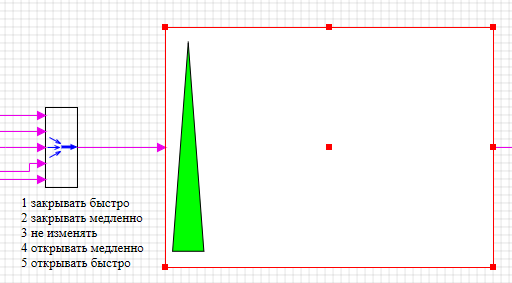


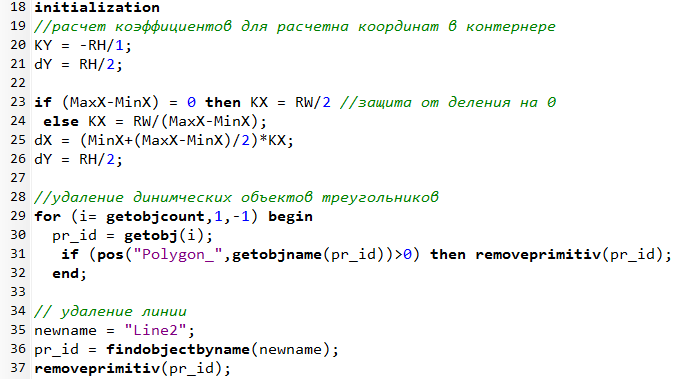
Рис. 3.7. Блок с треугольной функцией

Если теперь менять значения a[1], b[1], с[1] в параметрах блока, то каждый раз будет рисоваться новый треугольник. Вернемся в графический редактор блока и удалим все что мы нарисовали.

## 3.3 И моделируем, и рисуем

Завершим создания анимации блока работы нечеткой логики.

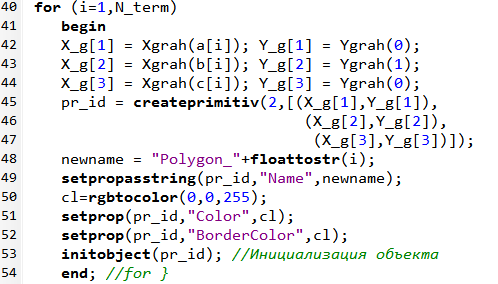
И начнем мы в секции инициализации, и начнем с того что сотрем все что еще не нарисовали, поскольку каждый раз при инициализации и изменений параметров будут создаваться новые полигоны, нужно удалить старые что бы не загромождать поле:



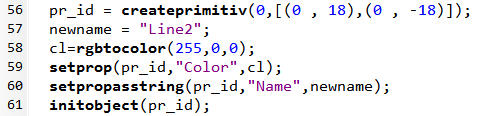
Данный кусок кода обратным счетом обходит все объекты и проверяет их имена, если имя содержит строку «**Polygon\_**», то мы удаляем это объект. Так же в этом куске кода мы удаляем линию, нарисуем ее позднее. Теперь наверняка все чисто и можно нарисовать столько термов, сколько задано в параметрах.

Создавая новые полигоны, мы присваиваем им имена начинающиеся на строку «*Polygon\_»,* это позволит нам в дальнейшем найти и удалить их при следующей инициализации.

Поскольку мне не направиться зеленый цвет, я в коде меняю его на синий, и убираю линию границы полигона, мне она тоже не нравиться в итоге у меня получается следующий код:



Полигоны и треугольные функции – это хорошо, но я хочу еще получить на схеме оценку выходной функции. Для этого я создаю линию, которая покажет, где между максимум и минимум находиться результат нечеткого вывода:



На этом создание изображения блока закончено, закроем секцию инициализации словом **end;**

Если сейчас сохранить скрипт и закрыть окно подтвердив сохранения изображения, то изображение на схеме будет отображать наглядно параметры фазификации блока нечеткого вывода (см. Рис. 3.8).

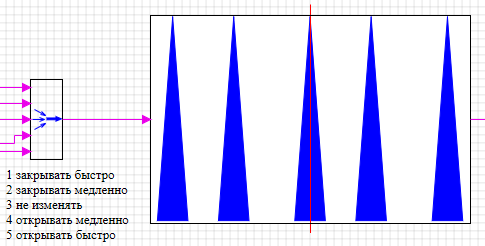
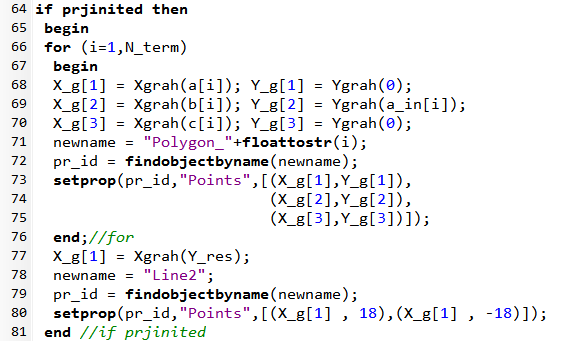


Рис. 3.8. Изображение блока фазификации

Ну и последний штрих оживление этих треугольников в процессе расчета. Текст см. ниже.

Все просто, если у нас проект находиться в состоянии расчёта, мы для каждого терма готовим массив точек треугольника, учитывая значения входных функций, тут-то и пригодилась переменная a\_in – она определяет высоту треугольника. По имени находим соответствующий полигон и устанавливаем ему новые точки.

С линией поступаем еще проще пересчитываем, координаты по результату вывода Xgraph(Y\_res) и передвигаем ее в нужное положение. Если вы все сделали правильно, то ваш блок во время расчета отображает состояние нечеткого вывода. Что бы это увидеть установите синхронизацию в параметрах расчета (кнопочка  в верху схемы, справа) с реальным времени и шаг вывода результата 0,005 (иначе расчет проскакивает так быстро что ничего не успеть увидеть)



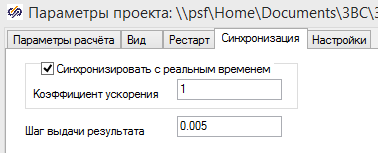
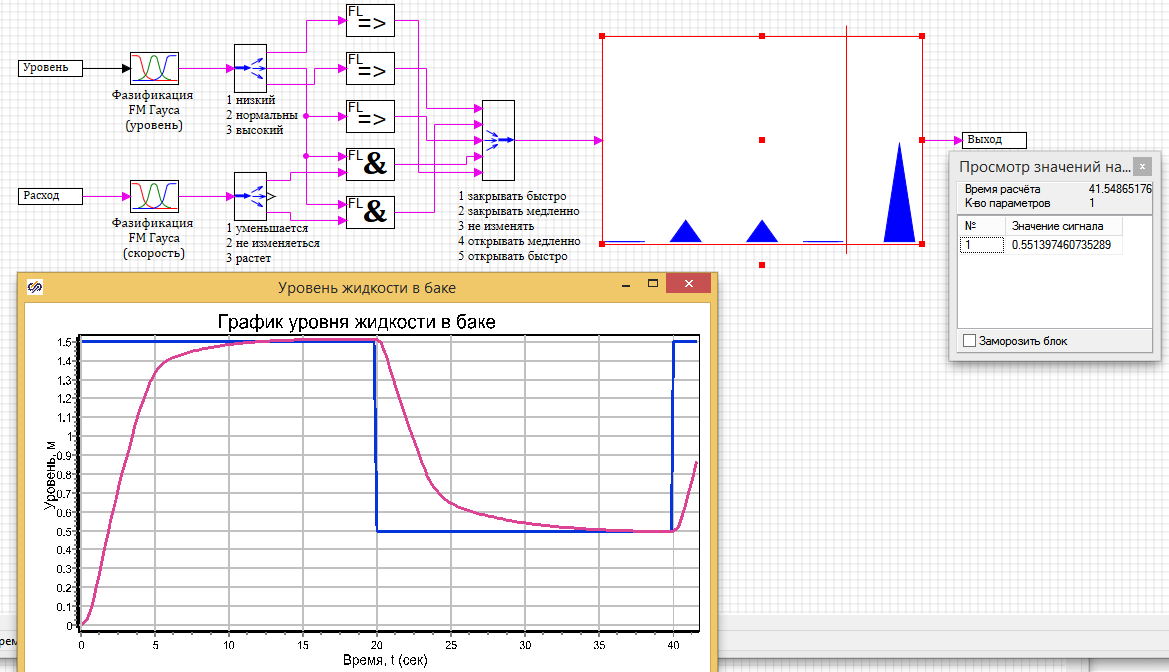


Рис. 3.9. Настройка синхронизации с реальным временем



Поздравляю вы теперь не только создали блок нечеткого вывода, но и создали наглядную анимацию работы этого блока непосредственно на схеме, в процессе расчета и подготовки данных.

Так же наглядно показано как просто создавать новые крутые блоки в SimInTech.

Но и это еще не все. Далее мы покажем, как создать код Си из нашего блока.

# 4 ГЕНЕРАТОР КОДА НА ЯЗЫКЕ СИ

## 4.1 Что это и зачем это нужно?

Дело в том, что SimInTech, хоть и работает под Linux, но является системой дорогой, сложной идеально предназначенной для моделирования и научного анализа моделей сложных систем и алгоритмов управления ими. Инструмент высоко-интеллектуальных ученых. А у практиков-инженеров напрашивается идея, если мы так классно и быстро слепили алгоритм управления, в нельзя ли его так же и запихнуть в систему управления, прямо из SimInTech? С радостью отвечаю на этот вопрос – можно!

Схема алгоритма, созданная в среде SimInTech может быть преобразована в код на языке Си. А уже код может быть скомпилирован под любую операционную систему или целевую платформу, для которой существует компиляторы. Таки образом мы можем отлаженную схему превратить в программу, которая работает везде, где можно запускать программы скомпилированные из языка Си. Наша схема может быть преобразована в исполняемый кода или динамическую библиотеку и запущена:

* в составе среды SimInTech;
* в совместно или в составе сторонней моделирующей программы;
* контроллерах реальной системы управления.

Именно так создают программы для управления атомными реакторами, сначала рисуют схемы в SimInTech, а потом, по нажатию волшебной кнопки, генерируют код Си, компилируют ее в программу и заливают в контроллер управления. Генератор Си в среде SimInTech сертифицирован, для генерации кода в системах важных для безопасности АЭС.

С другой стороны, если вдруг вы создали что-то невероятно умное и полезное, и не хотите, чтобы вашу внутреннюю структуру подсмотрели пользователи, вы можете преобразовать ваши модели в код Cи, собрать библиотеку для SimInTech и поставлять их заказчикам в виде dll, что бы они не раскрыли ваши технологические секреты.

И, наконец, третья причина использования кода Cи это ускорение расчета в 20 – 200 раз, в зависимости от типа модели.

## 4.2 Настраиваем генерацию кода для новых блоков

В SimInTech из расчетной схемы можно создать код Си, если схема создана из блоков в стандартной библиотеке блоков, то Си получается просто нажатием кнопки. Для блока, созданного своими руками на языке программирования есть как минимум два способа:

1. просто написать текст Си и разложить в поля свойств блока «*Язык программирования*». (см. Рис 4.1)
2. Указать **Да** для свойства «*Транслировать в код автоматически*»

Именно поэтому выполняя пункт 2.3 мы снесли все свойства оставив только одно, если снесли, то вернуть его обратно, можно путем копирования у блока «Язык программирования» (см. Рис 4.1)

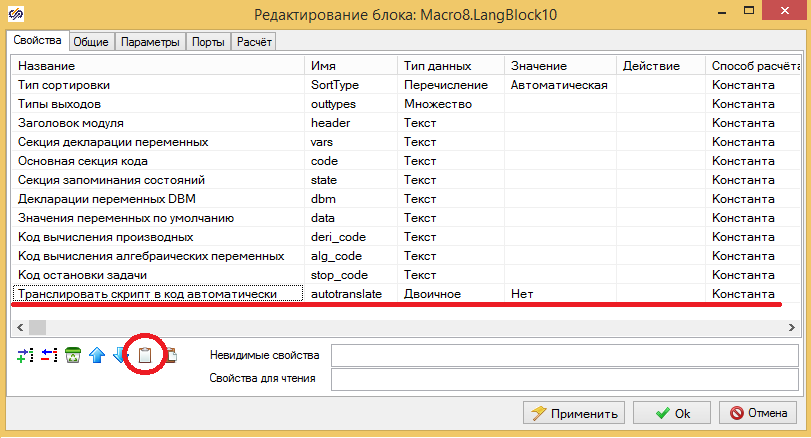


Рис. 4.1. Свойства блока отвечающие за генерацию кода

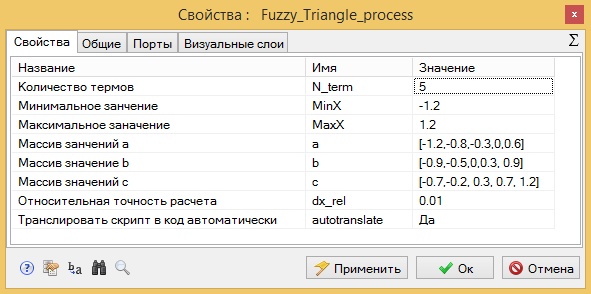


Рис. 4.2. Свойства блока нечеткого вывода с добавленным пунктом по генерации кода

Установите для этого значения «Да» для всех блоков, на схеме созданных с помощью языка программирования. Убедившись, что у все наших новых блоков, созданных с использованием языка программирования есть «*Транслировать скрипт в код автоматически*» и оно установлено в «Да» можно перейти к генерации кода.

## 4.3 Генерация кода Си или программист не нужен

Для создания кода, мы будем использовать внутренности субмодели «Блок управления на базе нечеткой логики», войдите в него выделите все блоки и скопируйте их в буфер обмена.

Создайте новый проекта (Меню «*Файл*»/ «*Схема автоматики*»), вставьте в него скопированные блоки и сохраните под именем «fl\_controller.prt», в той же папке где лежит полная модель. Все что у нас на данной схеме будет преобразовано в код Си и помещено в dll, для того чтобы эта dll могла работать в состав схемы необходимо объяснить ей, что у нас вход, а что выход, нужно заменить входы и выходы.

Необходимо заменить блоки «*Порты входа*» «***Уровень***» и «***Расход***» на блоки «*входной контакт S3*», расположенные в закладке «*Данные*», а блок «Порт выхода» «Выход» заменить на блок «*выходной контакт S3*». В свойствах контактов нужно указать ключевые слова «***input:0***» для первого входа (уровень) и «**input:1**» для второго входа (расход). Для выходного контакт имя должно быть указано как «***out:0***», так же в свойствах блока нужно указать тип контакта – «**float**» (см. Рис. 4.3)

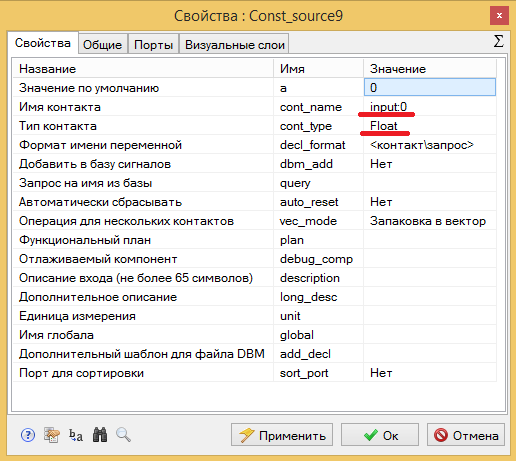


Рис. 4.3. Настройка контакта в будущей dll

В результате доработки схема должна выглядеть как рисунке 4.4

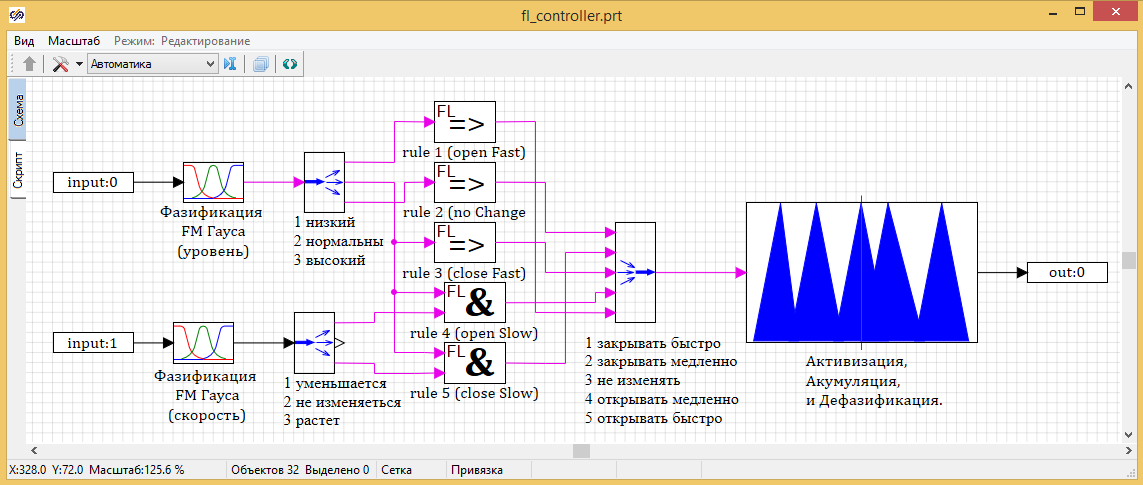


Рис. 4.4. Схема алгоритма подготовленная для генерации кода Си

Определим название для нашей будущей dll. Вызовем параметры расчета схемы (кнопочка  в верху схемы, справа)(см. Рис. 4.4), и зададим имя алгоритма как «***fl\_controller***» (cм. Рис. 4.5)

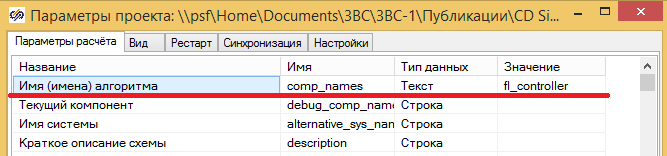


Рис. 4.5 настройка имени dll

Перейдем к настройкам генератора кода. В главном меню программы выбираем пункт «*Кодогенератор*», на закладке «*Настройки*», выбираем «Директория шаблонов кода», WinGW\_DLL, если у вас 32 разрядный SimInTech или WinGW\_DLL\_x64, если у вас 64 разрядный SimInTech. (см. Рис 4.6)

Если вы используете Linux или, не приведи господь QNX, вы лучше меня знаете что там и про что.

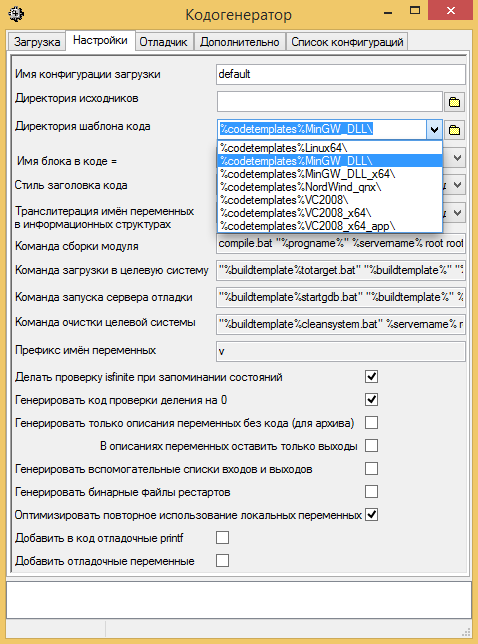


Рис. 4.6 Настройка кодогенератора

Сохраняем проект вы главном меню пункт «*Инструменты*» выбираем подпункт «Сгенерировать программу». И скорее всего вы увидите вот такое окно (см. рис. 4.7), где вам сообщат, что у вас не установлены компилятор MinGW, и добрые разработчики SimInTech специально для вас выложили ссылку, откуда вы можете его скачать.

Но несмотря, на то что компилятора нет, сам код сгенерировался и вы можете найти его в папке проекта. Пока качается компилятор, вы можете посмотреть как выглядит ваш проект записанный в коде Cи и даже найти там блоки, которые мы создали. (см. рис. 4.8)

Только что, вы сгенерировали код Си сертифицированный для использования в системах управления важных для безопасности АЭС.

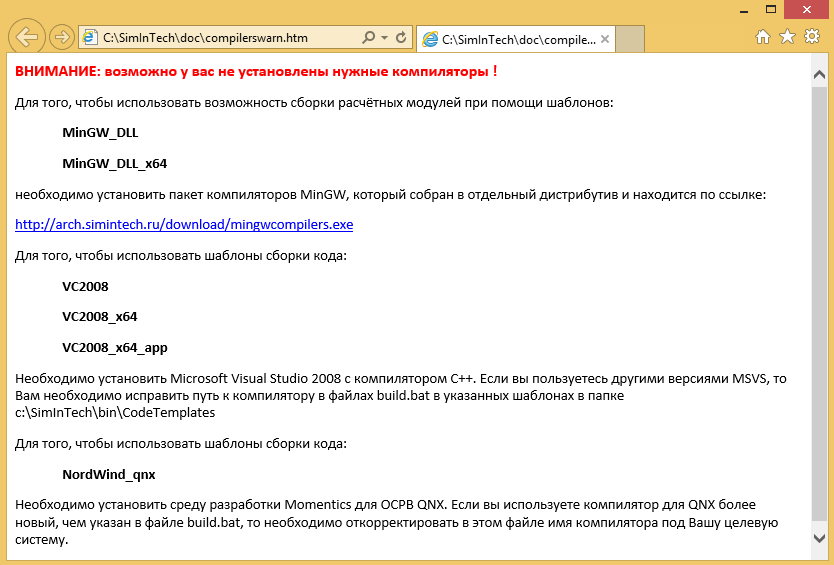


Рис. 4.7 Окно с ссылками для загрузки компилятора

|  |  |
| --- | --- |
|  |  |
| Рис. 4.8 Код, сгенерированный из SimInTech | Рис. 4.9 Настройка блока «Внешняя dll» |

После установки компилятора, снова выберете в главном окне «*Инструменты*» - «Сгенерировать программу». Если все установлено правильно, в папке появиться готовый файл с именем fl\_controller.dll. Поздравляют вы только что сделали блок управления на базе нечеткой логики запрограммировали его на Си и собрали в динамическую библиотеку. Программист больше не нужен!

Что бы убедиться, что он работает именно так же войдите в общую модель и удалите субмодель «Блок управления на основе нечеткой логики», вместо него разместить блок «*Внешняя dll*» из закладки «*Субстуркутуры*». В настройках блока мы меняем следующие параметры (см. рис 4.9):

Количество портов – **2** – два входа, которые мы отправляли в схему из модели

Массив размерностей выходов **[1]** – у нас один выход с размерностью 1.

Имена загружаемых dll – **fl\_controller.dll** имя файла созданной автоматически dll.

Имена файлов проектов для отладки – **fl\_controller.prt** имя проекта из которого мы создавали dll.

Подключаем настроенный блок в модель так что бы получилась схема как на рис. 4.10. и сохраняем ее под другим именем.

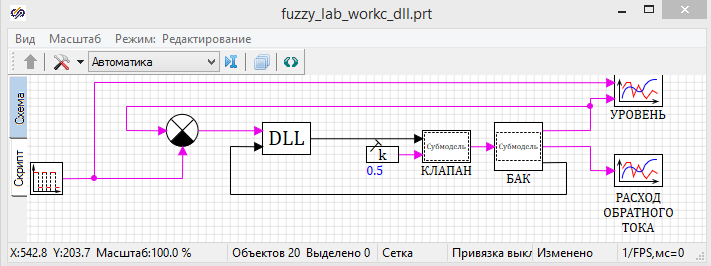


Рисунок 4.10 Схема модели с логикой управления подключенной как внешняя dll

Запуская модель на расчет мы можем убедиться, что работает система управления абсолютно так же как созданная в виде схемы. Если вы все делали согласно методическим указаниям, то все файлы находятся в одной директории, и при расчете в режиме синхронизации с реальным временем вы можете путем двойного клика на блоке DLL вызвать схему из которой был собрана dll наблюдать за моделированием, так же как и при стандартном моделировании схемы. При это вызываемая схема отображает значения как на линиях связи, может строить графики, а так же работает анимация блоков созданная ранее (см. Рис. 4.11).

Если же вы не передаете исходную схему моделирование происходит в режиме черного ящика.

Для сравнения скорости расчета в виде схемы и в виде скомпилированного кода снимите галочку «*Синхронизировать с реальным временем*» в параметрах расчета основной модели (кнопочка  в верху схемы, справа) (рис. 4.12).

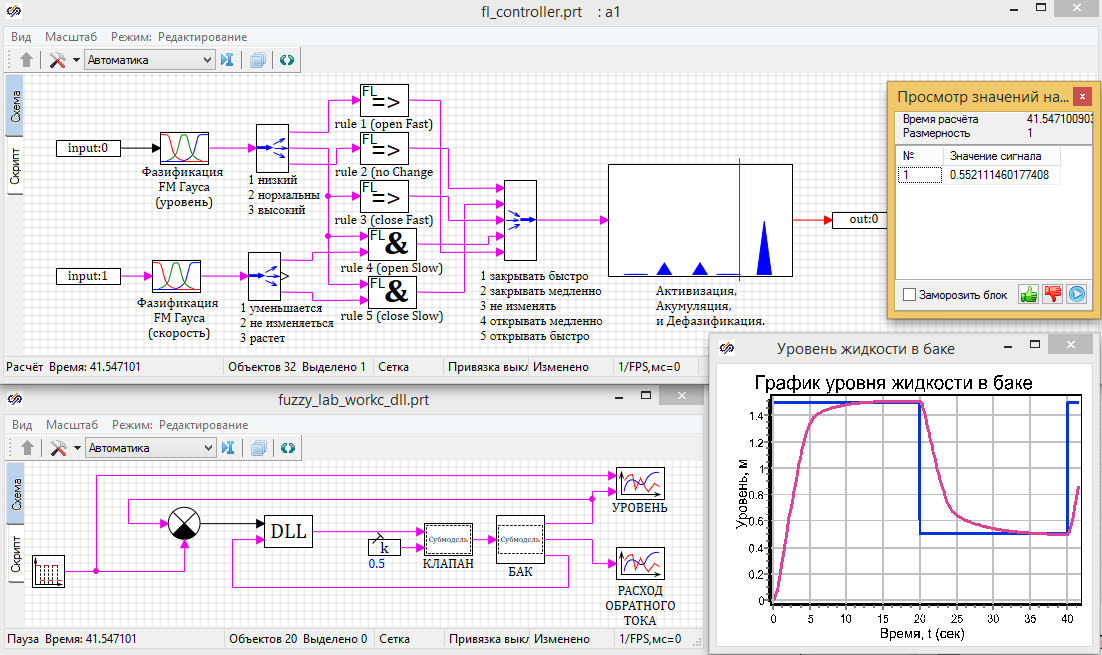


Рисунок 4.11 Отображения работы dll на схеме-источнике.

|  |  |
| --- | --- |
|  |  |
| Рис. 4.12 Отмена синхронизации при расчете модели | Рис. 4.13 Сравнение скорости расчет dll и схемы |

Запустите на расчет модель. Посчитает мгновенно, но если зайти в главное окно пункт меню «*Расчет*» подпункт «*Отладочная информация*», то коэффициент ускорения, покажет во сколько раз быстрее реального времени рассчитана модель. На рисунке 4.13 показана сравнение скорости модели нечеткой логики в виде схемы и в виде dll, в данном случае выигрыш dll составляет более чем в 200 раз.

# Заключение

Только что на ваших глазах, мы сделали набор блоков, который в Simulink выделен в отдельны ToolBox и продается за нормальные деньги. Данная лабораторная работа показывает, что с помощью инструмента SimInTech можно решить любые задачи для моделирования, упаковать модели в dll и продавать их как отдельные продукты. Удачи!