МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Набережночелнинский институт (филиал)

Кафедра Бизнес-информатики и математических методов в экономике

Визуальное моделирование экономических процессов в среде MATLAB
Часть 2. Разработка имитационных моделей средствами пакета SIMULINK

Учебно-методическое пособие

Набережные Челны
2019 г.

УДК 510.285
ББК 22.18

Печатается по решению учебно-методической комиссии экономического отделения Набережночелнинского института (филиала) федерального государственного автономного образовательного учреждения высшего образования «Казанский (Приволжский) федеральный университет», от «24» января 2019г. (протокол №5)

Рецензенты:
Доктор физ.-мат. наук, профессор А.Г. Исавнин
Доктор экономических наук, профессор А.Н. Макаров

Махмутов И.И., Шарипов Р.Ш. Визуальное моделирование экономических процессов в среде MATLAB. Часть 2. Разработка имитационных моделей средствами пакета SIMULINK: учебно-методическое пособие / И.И. Махмутов, Р.Ш. Шарипов. – Набережные Челны: Изд-во Набережночелнинского института КФУ, 2019. – 44 с.

Учебно-методическое пособие содержит последовательное изложение базовых понятий теории визуального моделирования экономических процессов в среде MATLAB. Разработка имитационных моделей средствами пакета SIMULINK. Подробно изложены: основные приемы работы в среде SIMULINK; библиотека SIMULINK; подсистемы; пример разработки имитационной модели.

Учебно-методическое пособие предназначено для использования в учебном процессе студентами технических направлений в экономике и экономического отделения дневной, заочной и дистанционной форм обучения.

Содержание

Введение 4
1. Основные приемы работы в среде SIMULINK 5
2. Библиотека SIMULINK 6
2.1. Раздел Sources (Источники) 6
Раздел Sinks (Получатели) 8
2.2. Раздел Math Operations 11
2.3. Раздел Discrete (Дискретные системы) 13
2.4. Раздел Discontinuities 13
2.5. Раздел Logic and Bit Operations 14
2.6. Раздел Signal Routing 14
2.7. Раздел User-Defined Functions 17
2.8. Раздел Commonly Used Blocks 18
Подсистемы 18
2.9. Создание подсистем 18
2.10. Виды подсистем 21
2.11. Маскирование подсистем 27
2.12. Справочные подсистемы 33
3. Пример разработки имитационной модели 33
4. Задания к лабораторным работам 37
5. Литература 42

Введение

Simulink является приложением к пакету MATLAB. При моделировании с использованием Simulink реализуется принцип визуального программирования, в соответствии с которым, пользователь на экране из библиотеки стандартных блоков создает модель устройства и осуществляет расчеты. При этом, в отличие от классических способов моделирования, пользователю не нужно досконально изучать язык программирования и численные методы математики, а достаточно общих знаний требующихся при работе на компьютере и, естественно, знаний той предметной области в которой он работает.

Simulink является достаточно самостоятельным инструментом MATLAB и при работе с ним совсем не требуется знать сам MATLAB и остальные его приложения. С другой стороны доступ к функциям MATLAB и другим его инструментам остается открытым и их можно использовать в Simulink. Часть входящих в состав пакетов имеет инструменты, встраиваемые в Simulink. Имеются также дополнительные библиотеки блоков для разных областей применения.

При работе с Simulink пользователь имеет возможность модернизировать библиотечные блоки, создавать свои собственные, а также составлять новые библиотеки блоков.

При моделировании пользователь может выбирать метод решения дифференциальных уравнений, а также способ изменения модельного времени (с фиксированным или переменным шагом). В ходе моделирования имеется возможность следить за процессами, происходящими в системе. Для этого используются специальные устройства наблюдения, входящие в состав библиотеки Simulink. Результаты моделирования могут быть представлены в виде графиков или таблиц.

Преимущество Simulink заключается также в том, что он позволяет пополнять библиотеки блоков с помощью подпрограмм написанных как на языке MATLAB, так и на языках C + +, Fortran и Ada.

1. Основные приемы работы в среде SIMULINK

Создать новый файл модели можно с помощью команды File/New/Model, или используя соответствующую кнопку на панели инструментов.

Чтобы расположить блоки в окне модели необходимо открыть соответствующий раздел библиотеки. Далее, указав курсором на требуемый блок и нажав на левую клавишу «мыши» - «перетащить» блок в созданное окно.

Для удаления блока необходимо выбрать блок а затем нажать клавишу **Delete** на клавиатуре.

Для изменения параметров блока необходимо дважды щелкнуть левой клавишей «мыши», указав курсором на изображение блока. Откроется окно редактирования параметров данного блока. При задании численных параметров следует иметь в виду, что в качестве десятичного разделителя должна использоваться точка, а не запятая.

После установки на схеме всех блоков из требуемых библиотек нужно выполнить соединение элементов схемы. Для соединения блоков необходимо указать курсором на «выход» блока, а затем, нажать и, не отпуская левую клавишу «мыши», провести линию к входу другого блока. После чего отпустить клавишу. В случае правильного соединения изображение стрелки на входе блока изменяет цвет. Для создания точки разветвления в соединительной линии нужно подвести курсор к предполагаемому узлу и, нажав правую клавишу «мыши», протянуть линию. Для удаления линии требуется выбрать линию (так же, как это выполняется для блока), а затем нажать клавишу **Delete** на клавиатуре.

Для повышения наглядности модели удобно использовать текстовые надписи. Для создания надписи нужно указать мышью место надписи и дважды щелкнуть левой клавишей мыши. После этого появится прямоугольная рамка с курсором ввода. Аналогичным образом можно изменить и подписи к блоками моделей.

2. Библиотека SIMULINK

2.1. Раздел Sources (Источники)

Блоки, входящие в раздел «Источники», предназначены для описания рабочей нагрузки моделируемой системы, а также для формирования сигналов, обеспечивающих управление работой модели в целом или отдельных ее частей.

Все блоки-источники имеют по одному выходу и не имеют входов.

Основные блоки раздела Sources:

1. Clock. Отображение и обеспечение времени моделирования. Обеспечивает представление текущего значения модельного времени на очередном шаге моделирования и используется при моделировании непрерывных систем. Имеет два параметра настройки.

- **Display Time** (Показать текущее время) – если флажок установлен, то внутри блока в числовой форме выводится текущее значение модельного времени.
- **Decimation** (дискретность) – целое число, определяющее периодичность формирования в блоке значения модельного времени.

2. Constant. Генерация постоянного значения. Данный блок генерирует определенное значение, независимое от времени. Блок генерирует один выход, который может быть скалярным или векторным. Пиктограмма блока отображает определенное значение или значения. Параметр блока:

- **Constant value** (значение константы) – параметр, определяющий значение сигнала. Значение сигнала может быть задано в виде числовой константы, в виде вектора (последовательности чисел, заключенных в квадратные скобки), в виде матрицы (последовательности векторов, заключенных в квадратные скобки), в виде вычисляемого выражения.

3. Digital Clock. Источник дискретного временного сигнала. Применяется в случаях, когда необходимо знать текущее время моделирования внутри дискретной системы. Его особенность состоит в том, что он не только формирует величину шага, но и вычисляет новое значение модельного времени, которое используется для проверки условия окончания

моделирования. Очередное значение модельного времени вычисляется как сумма предыдущего значения и величины шага моделирования.

4. **Random Number**. Генерирование нормально распределенных случайных чисел. Блок обеспечивает формирование сигналов, амплитуда которых является случайной величиной, распределенной по нормальному закону с заданными параметрами. Параметры блока:

- **Mean** (среднее значение)
- **Variance** (дисперсия)
- **Initial seed** (начальное значение) – задает начальное значение для инициализации генератора последовательности случайных чисел. При фиксированном значении этого параметра генератор всегда вырабатывает одну и ту же последовательность случайных чисел.

5. **Uniform Random Number**. Генерирование однородно распределенных случайных чисел. Блок однородного случайного числа генерирует однородно распределенные случайные числа на определенном интервале с определенным начальным значением. Значение сбрасывается каждый раз при начале моделирования. Сгенерированная последовательность является повторяющейся и может быть произведена любым блоком однородного случайного числа (**Uniform Random Number block**) с тем же начальным значением и параметрами. Параметры:

- **Minimum** (минимальное значение)
- **Maximum** (максимальное значение)
- **Initial seed** (начальное значение) – задает начальное значение для инициализации генератора последовательности случайных чисел

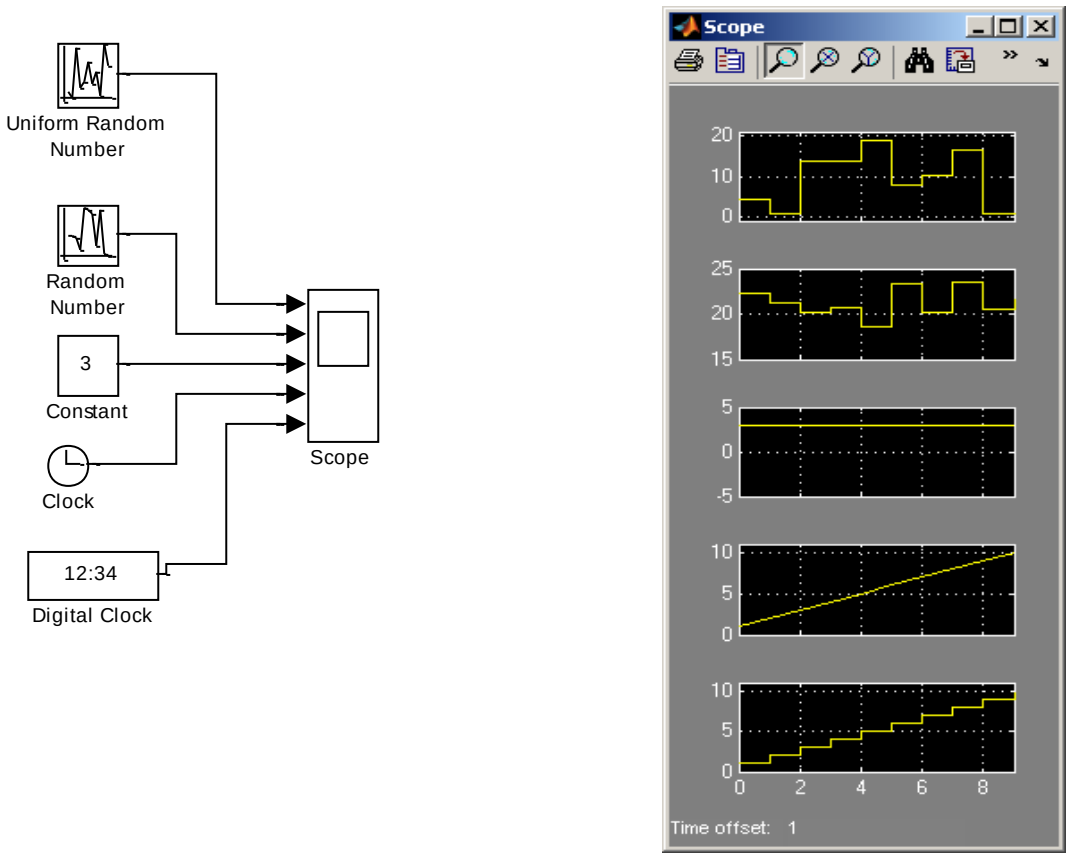

Рис 1. Демонстрация возможностей блоков раздела Sources

Раздел Sinks (Получатели)

Условно блоки данного раздела можно разделить на 3 вида:

- Блоки, используемые при моделировании в качестве «смотровых окон»: **Scope**, **XYGraph**, **Display**.
- Блоки, обеспечивающие сохранение промежуточных или входных результатов моделирования: **To File**, **To Workspace.**
- Блок управления моделированием: **Stop Simulation.**

Блоки раздела Sinks:

1. **Scope**. Отображение сигналов, генерируемых во время моделирования. Экран отображает свой вход как функцию от времени моделирования. Блок принимает один вход и может отобразить до 30 сигналов. Экран (Scope block) позволяет контролировать количество времени и область отображаемых входных значений. Если сигнал непрерывный, экран (Scope block) выводит

график, нарисованный по точкам. Если сигнал дискретный, блок выводит ступенчатый график. Блок позволяет в процессе моделирования наблюдать динамику изменения интересующих исследователя характеристик системы. По оси ординат шкалы измерений откладываются значения наблюдаемой величины, по оси абсцисс – значения модельного времени.

2. **Display**. Отображение значения входа. Предназначен для вывода на экран численных значений величин, фигурирующих в модели. Имеет 4 параметра настройки:

- Раскрывающийся список **Format** позволяет выбрать форматы вывода результата.
- Поле **Decimation** определяет периодичность вывода значений в окне.
- Флаг **Floating display –** Позволяет выбрать способ использования блока в модели. Если флажок установлен, то блок используется как свободный (т.е. без входного порта). Чтобы указать сигнал, отображаемый в окне Display, следует выделить соответствующую линию связи.
- Поле **Sample Time –** задает величину шага модельного времени, т.е. дискретность вывода данных в окне Display.

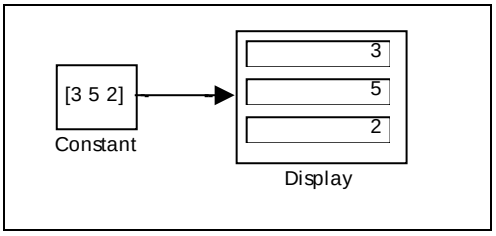

Рис 2. Вывод блоком Display векторных величин

3. **Stop Simulation**. Прекращение моделирования при ненулевом входе. Блок останова моделирования (Stop Simulation block) прекращает моделирование при ненулевом входе. Моделирование заканчивает текущий временной интервал прежде, чем остановиться. Если вход блока векторный, любой ненулевой элемент вектора вызывает останов моделирования. Вы

можете использовать этот блок в соединении с оператором отношения (Relational Operator block) для контроля, когда останавливается моделирование.

4. **To File**. Блок обеспечивает запись в MAT-файл данных, полученных в ходе моделирования. Блок имеет следующие параметры настройки:

- **Filename** (имя файла) – имя MAT-файла, в который будут записываться данные;
- **Variable name** (имя переменной) – имя переменной, по которому можно обращаться к данным, записанным в файл;
- **Decimation** (дискретность) – дискретность записи в файл. Параметр может принимать только целочисленные значения.
- **Sample Time** – величина шага моделирования для данного блока.

5. **XY Graph**. Отображение графика сигналов с помощью окна рисунков MATLAB. Первый вход предназначен для ввода аргумента, второй – для ввода значений функций. Блок обеспечивает построение двумерных графиков зависимостей произвольных величин, фигурирующих в модели. Блок имеет два входа, первый из которых предназначен для ввода аргумента, второй – для ввода значений функции этого аргумента.

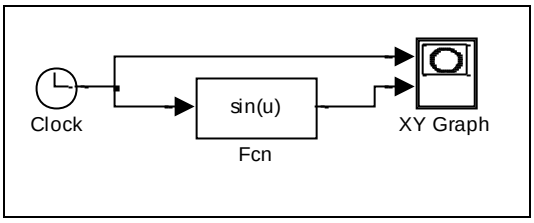

Рис 3. Построение графика функции sin(x)

6. **To Workspace**. Запись данных в матрицу рабочей области. Блок **To Workspace** записывает свой вход в определенную матрицу рабочей области. Блок записывает свой вход построчно, при этом каждая строка содержит все элементы входного вектора на временном шаге. Если матрица уже существует, ее содержимое перезаписывается. Параметры блока:

- **Variable name** – имя, под которым данные сохраняются в рабочей области.

- **Limit data points to last** – предельно допустимое число шагов моделирования, для которого регистрируются данные (по умолчанию – inf – данные регистрируются на всем интервале моделирования).
- **Decimation** – дискретность регистрации данных
- **Simple Time** – величина шага (дискретность изменения) модельного времени.
- **Save format** – формат, в котором будут записаны данные в рабочей области. Данный параметр имеет три возможных значения: Structure (в виде структуры), Structure with time (структура с полем времени), Array (массив).

Для просмотра записанных данных в рабочей области MATLAB необходимо ввести имя переменной, указанной в поле **Variable name**.

2.2. Раздел Math Operations

Данный раздел содержит блоки, которые реализуют элементарные алгебраические и тригонометрические функции.

1. **Abs**. Вывод абсолютного значения входа. Данный блок генерирует в качестве выхода абсолютное значение входа. Блок принимает один вход и генерирует один выход.

2. **Algebraic Constraint**. Ограничение входного сигнала до нуля. Данный блок ограничивает входной сигнал **f(*z*)** до нуля и выводит алгебраическое состояние **z**. Блок выводит значение, необходимое для того, чтобы произвести нуль на входе. Выход должен влиять на вход через какую-нибудь обратную связь. Параметр

- **Initial guess** – начальное приближение

3. **Dot Product**. Генерирование произведения. Блок генерирует произведение двух векторов.

4. **Gain**. Усиление входа блока. Усилитель (**Gain block**) генерирует свой выход путем умножения входа на определенную константу, переменную или выражение.

5. **Math Function**. Выполнение математической функции. Блок математической функции (**Math Function block**) выполняет многочисленные математические функции. Название функции отображается на пиктограмме блока. SIMULINK автоматически рисует нужное число портов.

6. **MinMax**. Вывод минимального или максимального входного значения. Блок минимакса (MiniMax block) выводит или минимальный, или максимальный элемент или элементы входа (входов). Можно выбрать, какую функцию применять, выбрав ее из списка параметра **Function.** Если блок содержит один входной порт, то он выводит скаляр, т.е., минимальный ли максимальный элемент входного вектора. Если блок содержит несколько входных портов, то он выполняет поэлементное сравнение входных векторов. Каждый элемент выходного вектора блока является результатом сравнения входных векторов.

7. **Product**. Генерирование произведения или частного входов блока. Блок произведения (**Product block**) умножает или делит входы блока в зависимости от значения параметра **Number of inputs:** Если значение представляет собой комбинацию символов « * » и « / », число входов блока равно числу символов. Пиктограмма блока показывает соответствующий символ рядом с каждым входным портом. Выход блока равен произведению всех входов, отмеченных « * », деленному на все входы, отмеченные « / ».

8. **Rounding Function**. Выполнение функции округления. Блок функции округления (**Rounding Function block**) выполняет общие функции округления.

9. **Sign.** Отображение знака входа. Блок знака (**Sign block**) показывает знак входа: 1, если вход больше нуля, 0 – если вход равен нулю, -1 – если вход меньше нуля.

10. **Sum**. Генерирование суммы входов. Сумматор (**Sum block**) складывает скалярные и/или векторные входы, или элементы единичного векторного входа, в зависимости от числа входов блока. Если у блока более одного входа, выход блока равен поэлементной сумме входов. Если все входы

скалярные, выход тоже скаляр. Если у блока один векторный вход, выход блока скалярная сумма элементов входа.

11. **Trigonometric Function**. Блок обеспечивает выполнение тригонометрической функции. Блок тригонометрической функции (**Trigonometric Function block**) вычисляет общие тригонометрические функции. Можно выбрать одну из этих функций из списка **Function**: sin, cos, tan, asin, acos, atan, atan2, sinh, cosh, tanh. Выход блока – результат вычисления функции над входом или входами. На пиктограмме блока появляется имя функции. SIMULINK автоматически рисует необходимое число входных портов.

2.3. Раздел Discrete (Дискретные системы)

В данный раздел входят блоки, с помощью которых в модели может быть описано поведение дискретных систем.

Блоки раздела Discrete:

1. **Memory**. Вывод входа блока на предыдущем шаге интегрирования. Блок памяти (Memory block) выводит свой вход на предыдущем интервале времени.

2.4. Раздел Discontinuities

Данный раздел содержит достаточно универсальные блоки, которые могут быть использованы при построении моделей систем различных типов.

Блоки раздела:

1. **Dead Zone**. Обеспечение области нулевого выхода. Блок мертвой зоны (Dead Zone block) генерирует нулевой выход внутри определенной области, называемой его мертвой зоной. Нижний и верхний пределы мертвой зоны определяются как параметры **Start of dead zone** и **End of dead zone**. Выход блока зависит от входа мертвой зоны. Если вход находится внутри мертвой зоны (больше чем нижний предел и меньше чем верхний предел), выход равен нулю. Если вход больше чем или равен верхнему пределу, выход равен входу

минус верхний предел. Если вход меньше чем или равен нижнему пределу, выход равен входу минус нижний предел. Если нижний и верхний пределы равны, входной сигнал проходит через блок и выводится как выход

2. **Relay.** Переключение выхода между двумя константами. Реле (Relay block) позволяет переключать выход между двумя определенными значениями. Реле остается включенным до тех пор, пока значение входа не опустится ниже значения параметра **Switch off point**. Реле остается выключенным до тех пор, пока вход не достигнет значения параметра **Switch on point**. Блок принимает один вход и генерирует один выход.

3. **Saturation.** Ограничение диапазона сигнала. Блок насыщения (Saturation block) накладывает верхнюю и нижнюю границу на сигнал. Когда входной сигнал находится внутри диапазона, определенного параметрами **Lower limit** и **Upper limit**, то он проходит через блок без изменений. Когда входной сигнал выходит за эти границы, то он отсекается верхней или нижней границей

2.5. Раздел Logic and Bit Operations

Блоки:

1. **Relational Operator**. Выполнение определенной операции отношения. Оператор отношения (Relational Operator block) выполняет операцию отношения с двумя входами и производит соответствующий выход.

2.6. Раздел Signal Routing

Данный раздел блоков является, пожалуй, самым полезным для имитационного моделирования.

Блоки:

1. **Demux**. Разделение векторного сигнала на выходные сигналы. Демультиплексор (Demux block) разделяет векторный входной сигнал на выходные линии, каждая из которых может содержать скалярный или векторный сигнал. Если параметр **Number of outputs** скалярный, блок

разделяет входной сигнал на данное число выходных сигналов. Ширина выходных сигналов зависит от ширины входного вектора и количества выходов. Если ширина входного сигнала равна количеству выходов, блок разделяет вектор входного сигнала на скалярные сигналы. Если ширина входного сигнала делится нацело на число выходов, блок разделяет входной сигнал на векторные сигналы равной ширины. Если ширина входного сигнала не делится нацело на число выходов (и они не одинаковы), блок разделяет входной сигнал на векторные сигналы неравной ширины, и SIMULINK выдает предупреждение.

2. **From**. Прием входа от блока перехода (Goto block). Блок Из (From block) принимает сигнал от соответствующего ему блока перехода (Goto block), затем выводит его в качестве выхода. Блоки Из и перехода позволяют передавать сигнал от одного блока к другому без реального соединения этих блоков. Каждый блок Из (From block) связан с блоком перехода (Goto block). Вход блока перехода (Goto block) передается блоку Из (From block), который затем передает его блоку, соединенному с ним. Чтобы связать блок перехода (Goto block) с блоком Из (From block), введите признак блока перехода (Goto block) как параметр **Goto tag.** Блок Из (From block) может получать сигнал только от одного блока перехода (Goto block), а блок перехода (Goto block) может передавать свой сигнал нескольким блокам Из (From block).

3. **Goto**. Передача входа блока блокам Из (From block). Блок перехода (Goto block) передает свой вход соответствующим блокам Из (From block). Блоки Из (From block) и перехода (Goto block) позволяют передавать сигнал от одного блока к другому без реального соединения между ними. Блок перехода (Goto block) может передавать свой входной сигнал нескольким блокам Из (From block), однако блок Из (From block) может получать сигнал только от одного блока перехода (Goto block). Вход блока перехода (Goto block), с которым связаны блоки Из (From block), передается так, как будто блоки соединены физически. Об ограничениях на использование блоков Из (From block) и перехода (Goto block) см. описание блока Из (From block) на странице

9–59. Блоки перехода (Goto block) и Из (From block) сочетаются по использованию признаков перехода, определенных как параметр **Tag.**

4. **Manual Switch**. Переключение между двумя входами. Ручной переключатель (Manual Switch block) – это ключ, который выбирает один из двух входов, через который будет происходить передача на выход. Чтобы переключиться между входами, дважды щелкните на пиктограмме блока (диалогового окна не существует). Выбранный вход передается на выход в то время, как невыбранный вход игнорируется. Вы можете установить переключатель до начала моделирования или переключить его во время моделирования для интерактивного управления ходом сигнала. Ручной переключатель (Manual Switch block) принимает все типы сигналов и сохраняет свое текущее состояние при сохранении модели.

5. **Multiport Switch**. Выбор между входами блока. Блок многопортового переключения (Multiport Switch) выбирает один вход среди нескольких. Первый (верхний) вход – это управляющий вход, а другие входы – входы переключения. Значение управляющего входа определяет, какой вход переключения пройдет через выходной порт. SIMULINK округляет значение управляющего входа до ближайшего (положительного) целого и пропускает соответствующий этому значению вход переключения.

6. **Mux**. Объединение нескольких входных линий в векторную линию. Мультиплексор (Mux block) объединяет нескольких входных линий в одну векторную линию. Каждая входная линия может содержать скалярный или векторный сигнал. Выход мультиплексора (Mux block) – вектор. Если вы определите параметр **Number of inputs** как скаляр, SIMULINK определит ширину входов путем проверки выходных портов блоков, подающих входные сигналы на мультиплексор (Mux block). Если какой-нибудь вход – вектор, все его элементы объединяются блоком.

7. **Selector**. Выбор входных элементов. Селектор (Selector block) генерирует в качестве выхода выбранные элементы входного вектора. Параметр **Elements** определяет порядок элементов входного вектора в

выходном векторе. Параметр должен быть определен как вектор, иначе будет выбран только один элемент. Пиктограмма блока показывает порядок элементов входного вектора графически. Если блок достаточно небольшой, на пиктограмме отображается имя блока.

8. **Switch**. Переключение между двумя входами. Ключ (Switch block) пропускает один из двух входов на выход в зависимости от значения третьего входа, называемого управляющим входом. Если сигнал на управляющем (втором) входе больше чем или равен параметру **Threshold**, блок пропускает первый вход; иначе, он пропускает третий вход.

2.7. Раздел User-Defined Functions

Блоки:

1. **Fcn.** Применение определенного выражения к входу. Блок функции (Fcn) применяет определенное выражение языка C к входу. Выражение может быть составлено из одного или нескольких следующих компонентов:

- u – вход блока. Если u – вектор, u(i) представляет собой i-ый элемент вектора; u(1) или просто u представляет собой первый элемент
- Числовые константы
- Арифметические операторы (+ – * /)
- Операторы отношений (= = != > < >= <=). Выражение возвращает 1, если отношение истинно; иначе возвращает 0.
- Логические операторы (&& || !). Выражение возвращает 1, если если отношение истинно; иначе возвращает 0.
- Скобки
- Математические функции – abs, acos, asin, atan, atan2, ceil, cos, cosh, exp, fabs, floor, hypot, ln, log, log10, pow, power, rem, sign, sin, sinh, sqrt, tan, tanh.
- Переменные рабочей области. Имена переменных, которые не распознаются в списке переменных, передаются для оценки в MATLAB. Элементы векторов и матриц должны быть обозначены специально (например, A(1,1), вместо A для первого элемента матрицы).

2. **MATLAB Fcn.** Применение функции или выражения MATLAB к входу. Блок функции MATLAB (MATLAB Fcn block) применяет определенную функцию или выражение MATLAB к входу. Блок принимает один вход и генерирует один выход. К входу применяется определенная функция или выражение. Выход функции должен соответствовать ширине выхода блока, иначе возникнет ошибка. Параметр **MATLAB function**

2.8. Раздел Commonly Used Blocks

Блоки:

1. **Ground.** Заземление неподсоединенного входного порта. Земля (Ground block) используется для соединения блоков, чьи входные порты не подсоединены к другим блокам. Если вы запустите моделирование с блоками, имеющими неподсоединенные входные порты, SIMULINK выведет предупредительное сообщение. Использование земли (Ground block) для «заземления» таких блоков позволяет избежать предупредительных сообщений. Земля (Ground block) выводит сигнал с нулевым значением.

2. **Terminator**. Завершение неприсоединенного выходного порта. Наконечник (Terminator block) может быть использован для покрытия блоков, чьи выходные порты не соединены с другими блоками. Если вы запустите моделирование с блоками, имеющими неприсоединенные выходные порты, SIMULINK выдаст предупредительное сообщение. Использование наконечников (Terminator block) для покрытия таких блоков помогает избежать предупредительных сообщений.

Подсистемы

2.9. Создание подсистем

Подсистема это фрагмент **Simulink**-модели, оформленный в виде отдельного блока. Использование подсистем при составлении модели имеет следующие положительные стороны:

1. Уменьшает количество одновременно отображаемых блоков на экране, что облегчает восприятие модели (в идеале модель полностью должна отображаться на экране монитора).

2. Позволяет создавать и отлаживать фрагменты модели по отдельности, что повышает технологичность создания модели.

3. Позволяет создавать собственные библиотеки.

4. Дает возможность синхронизации параллельно работающих подсистем.

5. Позволяет включать в модель собственные справочные средства.

6. Дает возможность связывать подсистему с каким-либо **m**-файлом, обеспечивая запуск этого файла при открытии подсистемы (нестандартное открытие подсистемы).

Использование подсистем и механизма их блоков позволяет создавать блоки, не уступающие стандартным по своему оформлению (собственное окно параметров блока, пиктограмма, справка и т.п.).

Количество подсистем в модели не ограничено, кроме того подсистемы могут включать в себя другие подсистемы. Уровень вложенности подсистем друг в друга также не ограничен.

Связь подсистемы с моделью (или подсистемой верхнего уровня иерархии) выполняется с помощью входных (блок **Inport** библиотеки **Sources**) и выходных (блок **Outport** библиотеки **Sinks**) портов. Добавление в подсистему входного или выходного порта приводит к появлению на изображении подсистемы метки порта, с помощью которой внешние сигналы передаются внутрь подсистемы или выводятся в основную модель. Переименование блоков **Inport** или **Outport** позволяет изменить метки портов, отображаемые на пиктограмме подсистемы со стандартных (**In** и **Out**) на те, которые нужны пользователю.

Подсистемы могут быть виртуальными (**Subsystem**) и монолитными (**Atomic Subsystem**). Отличие этих видов подсистем заключается в порядке выполнения блоков во время расчета. Если подсистема является виртуальной,

то **Simulink** игнорирует наличие границ отделяющих такую подсистему от модели при определении порядка расчета блоков. Иными словами в виртуальной системе сначала могут быть рассчитаны выходные сигналы нескольких блоков, затем выполнен расчет блоков в основной модели, а затем вновь выполнен расчет блоков входящих в подсистему. Монолитная подсистема считается единым (неделимым) блоком и **Simulink** выполняет расчет всех блоков в такой подсистеме, не переключаясь на расчеты других блоков в основной модели. Изображение монолитной подсистемы имеет более толстую рамку по сравнению с виртуальной подсистемой.

Подсистемы могут быть также управляемыми или неуправляемыми. Управляемые подсистемы всегда являются монолитными. Управляемые подсистемы имеют дополнительные (управляющие) входы, на которые поступают сигналы, активизирующие данную подсистему. Управляющие входы расположены сверху или снизу подсистемы. Когда управляемая подсистема активизирована – она выполняет вычисления. В том случае если управляемая подсистема пассивна, то она не выполняет вычисления, а значения сигналов на ее выходах определяются настройками выходных портов.

Существует два способа создания подсистем:

1. Скопировать нужную подсистему из библиотеки **Subsystem** в модель.
2. Выделить с помощью мыши нужный фрагмент модели и выполнить команду **Create Subsystem** из меню **Edit** окна модели. Выделенный фрагмент будет помещен в подсистему, а входы и выходы подсистемы будут снабжены соответствующими портами. Данный способ позволяет создать виртуальную неуправляемую подсистему. В дальнейшем, если это необходимо, можно сделать подсистему монолитной, изменив ее параметры, или управляемой, добавив управляющий элемент из нужной подсистемы находящейся в библиотеке. Отменить группировку блоков в подсистему можно командой **Undo**.

Доступ к окну параметров подсистемы осуществляется через меню **Edit** командой **Subsystem Parameters…**

Параметры подсистем*:*

1. **Show port labels –** Показать метки портов.

2. **Treat as atomic unit** – Считать подсистему монолитной. Таким образом, блоки виртуальной и монолитной подсистем – это один и тот же блок, отличающийся значением данного параметра.

3. **Read/Write permissions –** Доступность подсистемы для изменений. Выбирается из списка:

- **ReadWrite –** Пользователь может открывать и изменять подсистему.
- **ReadOnly –** Пользователь может открывать подсистему только для просмотра.
- **NoReadOrWrite –** Пользователь не может открывать и изменять подсистему.

4. **Name of error callback function –** Имя функции используемой для обработки ошибок возникающих в данной подсистеме.

2.10. Виды подсистем

Управляемая уровнем сигнала подсистема Enabled Subsystem

Подсистема **Enabled Subsystem** (в дальнейшем **E**-подсистема) активизируется при наличии положительного сигнала на управляющем входе. Если входной сигнал векторный, то подсистема активизируется, если хотя бы один элемент принимает положительное значение. Величина выходного сигнала в том случае, если система заблокирована, определяется настройками выходных портов подсистемы (блоки **Outport**). В том случае если параметр **Output when disabled** (вид сигнала на выходе подсистемы) выходного порта имеет значение **held**, то выходной сигнал подсистемы равен последнему рассчитанному ею значению, если же этот параметр имеет значение **reset**, то выходной сигнал подсистемы равен значению, задаваемому параметром **Initial output** (начальное значение).

Свойства **E**-подсистемы определяются параметрами блока **Enable**, который может находиться в любом месте данной подсистемы.

Параметры подсистемы:

1. **States when enabling –** Состояние при запуске. Параметр задает состояние подсистемы при каждом запуске. Выбирается из списка:

- **held –** Использовать предыдущее состояние (последнее состояние, когда система была активна).
- **reset –** Использовать начальное (исходное) состояние.

2. **Show output port –** Показать выходной порт. При установленном флажке на пиктограмме блока **Enable** появляется дополнительный выходной порт, сигнал с которого может быть использован для управления блоками внутри подсистемы.

Управляемая фронтом сигнала подсистема Triggered Subsystem

Подсистема **Triggered Subsystem** (в дальнейшем **T**-подсистема) включается фронтом (перепадом уровня) управляющего сигнала и выполняет вычисления только на том шаге моделирования, где произошло это изменение. Если входной сигнал векторный, то подсистема активизируется, если хотя бы в одном элементе изменяется уровень сигнала. Возврат **T**-подсистемы в исходное состояние не производится (подсистема сохраняет последнее значение до следующего запуска), поэтому параметр **States when enabling** выходных портов имеет значение **held**, и недоступен для изменения.

В **T**-подсистеме могут использоваться блоки, для которых модельное время является наследуемым параметром от предыдущего блока (например, **Gain** или **Logical Operator**), а также дискретные блоки, для которых параметр **sample time** имеет значение **–1** (минус один).

Свойства **T**-подсистемы определяются параметрами блока **Trigger**, который может находиться в любом месте данной подсистемы. Его параметры перечислены ниже.

Параметры подсистемы:

1. **Trigger type –** Тип триггера. Выбирается из списка:

- **rising –** Активизация подсистемы положительным фронтом.
- **falling –** Активизация подсистемы отрицательным фронтом.

- **either –** Активизация подсистемы как положительным, так и отрицательным фронтом.
- **function-call –** Активизация подсистемы определяется логикой работы заданной **S**-функции.

2. **Show output port –** Показать выходной порт.

Управляемая уровнем и фронтом сигнала подсистема Enabled and Triggered Subsystem

Подсистема **Enabled and Triggered Subsystem** (в дальнейшем **ET**-подсистема) включается фронтом сигнала поступающего на **T**-вход системы при наличии положительного сигнала на **E**-входе системы. Так же как и **Triggered Subsystem** эта подсистема выполняет вычисления только на том шаге моделирования, где произошло изменение управляющего сигнала на **T**-входе. Параметр **States when enabling** блока **Enable** не оказывает влияния на работу **ET**-подсистемы.

Оба управляющих сигнала могут быть векторными.

Блок условного оператора If

Блок условного оператора обеспечивает формирование управляющих сигналов для подсистем **If Action Subsystem**. Блок является аналогом оператора **if-else** языка программирования **C**.

Параметры блока:

1. **Number of inputs –** Количество входов.
2. **If expression –** Условное выражение. Условное выражение может включать в себя следующие знаки: **<. <=, ==, ~=, >, >=, &, |, [ ],** а также унарный минус. Если записанное условное выражение истинно, то на выходном **If**-порту блока формируется управляющий сигнал.
3. **Elseif expressions –** Одно или список альтернативных условных выражений разделенных запятыми, вычисляющихся, если условное выражение **If expression** ложно. Каждому условному выражению, записанному в списке **Elseif expressions** соответствует выходной **Elseif**-порт на котором формируется

управляющий сигнал, если соответствующее условное выражение истинно. При этом алгоритм вычисления альтернативных условных выражений таков, что если одно из альтернативных условных выражений окажется истинным, то следующие в списке выражения не проверяются. Альтернативное условное выражение может включать в себя те же знаки, что и выражение **If expression**.

4. **Show else condition** (флажок) – Показать **Else**-порт. На **Else**-порту формируется управляющий сигнал, если условное выражение и все альтернативные условные выражения ложны.

На пиктограмме блока отображаются условные выражения, записанные в его параметрах. Добавление каждого нового альтернативного условного выражения приводит к появлению нового **Elseif** выходного порта.

Если входные сигналы блока являются скалярами, то для их обозначения в выражениях используется запись вида **u1**, **u2**, **u3** и т.д. Если входные сигналы векторные, то для обозначения элементов вектора используются выражения вида **u1(1)**, **u1(2)**, **u2(1), u2(2)** и т.д.

Блок переключателя Switch Case

Блок переключателя обеспечивает формирование управляющих сигналов для подсистем **Case Action Subsystem**. Блок является аналогом оператора **Switch** языка программирования **C**.

Параметры блока:

1. **Case conditions –** Список значений входных сигналов (целое число). Каждому значению соответствует отдельный выходной **Case**-порт. Если значение входного сигнала, поступающего на вход блока **Switch Case**, совпадает с каким либо значением из списка, то на соответствующем выходе блока формируется управляющий сигнал. Если входной сигнал не является целым, то его дробная часть отбрасывается. В выражении **Case conditions** можно использовать квадратные скобки, если необходимо вырабатывать управляющий сигнал на каком-либо порту для нескольких значений входного сигнала. Например, выражение **{1,[7,9]}** задает два выходных **Case**-порта. На первом из них управляющий сигнал формируется, если входной сигнал блока

равен **1**, а на втором, – если входной сигнал равен **7** или **9**. В выражении **Case conditions** можно использовать также диапазоны значений. Например, выражение **{1:5}** определяет, что для единственного выходного **Case**-порта выходной сигнал будет вырабатываться, если входной сигнал блока равен **1**, **2**, **3**, **4** или **5**.

2. **Show default case** (флажок) **–** Показать **default case**-порт. На выходе **default case**-порта формируется управляющий сигнал, если входной сигнал блока не совпадает ни с одним значением, перечисленным в списке **Case conditions**.

Управляемая по условию подсистема Action Subsystem

Подсистема предназначена для работы под управлением блоков **If** или **Switch Case**. В первом случае она называется **If Action Subsystem**, а во втором **Switch Case Action Subsystem**.

Параметры подсистемы определяются настройками ее выходных портов, а также настройкой блока **Action Port**, наличие которого в подсистеме и превращает ее в **Action Subsystem**.

Блок имеет один параметр настройки: **States when execution is resumed –** Состояние подсистемы системы при следующем возобновлении работы. Значение параметра выбирается из списка:

- **held –** Использовать предыдущее состояние (последнее состояние, когда система была активна).
- **reset –** Использовать начальное (исходное) состояние.

Рассматриваемый параметр оказывает такое же действие на поведение подсистемы как параметр **States when enabling** блока **Enable**.

Управляемая подсистема For Iterator Subsystem

Управляемая подсистема **For Iterator Subsystem** представляет собой подсистему, которая выполняется неоднократно в течение одного такта моделирования. Количество повторений должно быть известно заранее и может задаваться внешним источником сигнала или с помощью параметра блока.

Основные свойства подсистемы задает итерационный блок **For Iterator**. Блок является аналогом оператора цикла **For** языка программирования **C**.

Блок **For** может находиться в любом месте подсистемы. Его параметры:

1. **States when starting –** Состояние подсистемы при следующем запуске. Значение параметра выбирается из списка:

- **held –** Использовать предыдущее состояние (последнее состояние, когда система была активна).
- **reset –** Использовать начальное (исходное) состояние.

2. **Iteration limit source –** Источник, задающий тип итераций.

- **internal –** Внутренний.
- **external –** Внешний.

3. Iteration limit – Количество итераций. Параметр доступен, если выбран внутренний источник числа итераций.

Управляемая подсистема While Iterator Subsystem

Управляемая подсистема **While Iterator Subsystem** представляет собой подсистему, которая выполняется неоднократно в течение одного такта моделирования. Количество повторений заранее не известно. Цикл прекращается, если значение логического сигнала на управляющем входе подсистемы станет равно **FALSE**. Основные свойства подсистемы задает итерационный блок **While Iterator**. Блок является аналогом оператора цикла **while (do-while)** языка программирования **C**.

Свойства **While Iterator Subsystem** определяются параметрами блока **While Iterator.** Его параметры перечислены ниже.

Параметры подсистемы:

1. **Maximum number of iterations –** Максимальное количество итераций. Если значение параметра равно **-1** (минус один), то количество итераций не ограничивается.

1. **While loop type –** Тип цикла. Выбирается из списка:

- **while –** Цикл **while**.
- **do-while –** Цикл **do-while**.

2. **States when starting –** Состояние подсистемы системы при следующем запуске. Значение параметра выбирается из списка:

- **held –** Использовать предыдущее состояние (последнее состояние когда система была активна).
- **reset –** Использовать начальное (исходное) состояние.

3. **Show iteration number port –** Отобразить на пиктограмме блока выходной порт, с которого снимается сигнал номера итерации.

4. **Output data type –** Тип данных выходного сигнала порта. Значение параметра выбирается из списка: **int32**, **int16**, **int8** и **double**.

Входной порт **IC** позволяет задать начальное значение сигнала прекращающего выполнение цикла **while**. При использовании цикла **do-while** подсистема будет выполнена хотя бы один раз (поскольку проверка условия в этом случае производится в конце цикла).

2.11. Маскирование подсистем

Механизм маскирования подсистем позволяет оформить подсистему как полноценный библиотечный блок, т.е. снабдить подсистему собственным окном параметров, пиктограммой, справочной системой и т.п.

Маскирование подсистем дает пользователю следующие преимущества:

1. Расширяет возможности пользователя по управлению параметрами модели.
2. Позволяет создавать более понятный интерфейс подсистемы.
3. Повышает наглядность блок-диаграммы.
4. Расширяет возможности построения сложных моделей.
5. Повышает защищенность модели от несанкционированной модификации.

Для выполнения маскирования имеющейся подсистемы необходимо предварительно выполнить следующие действия:

1. Определить какие параметры подсистемы должны задаваться пользователем в будущем окне параметров. Задать эти параметры в подсистеме с помощью идентификаторов (имен).

2. Определить каким образом параметр должен задаваться в окне диалога (с помощью строки ввода, выбором из раскрывающегося списка или установкой флажка).

3. Разработать эскиз пиктограммы блока.

4. Создать комментарии (справку) по использованию подсистемы.

Маскирование подсистемы выполняется с помощью **Mask Editor** (редактор маски). Для запуска редактора маски необходимо выделить маскируемую подсистему и выполнить команду **Mask Subsystem…** из меню **Edit**. Можно также воспользоваться контекстным меню.

После того как маскирование системы будет выполнено, двойной щелчок на ее изображении будет открывать окно параметров подсистемы, а не окно модели. Открыть саму подсистему (окно модели) для редактирования или просмотра можно командой **Look under mask** из меню **Edit** или контекстного меню.

Создание окна параметров

Окно параметров создается с помощью вкладки **Parameters** (Инициализация) редактора маски. Для создания поля ввода параметра с его описанием необходимо выполнить следующие действия:

1. Нажать кнопку **Add** (Добавить).

2. Ввести описание параметра в поле **Prompt** (Подсказка). В качестве описания параметра обычно используется его название в виде текста, например, “**Gain**”, “**Constant value**” и т.п.

3. Указать идентификатор параметра в поле **Variable** (Переменная). Естественно, что это должен быть один из тех идентификаторов, который использовался при задании параметров блоков внутри подсистемы (хотя это не обязательно, поскольку параметр может быть использован и для модификации самого окна диалога). Все переменные, идентификаторы которых заданы на

вкладке **Parameters**, помещаются в **Mask Workspace** – локальную рабочую область маски и являются доступными только внутри подсистемы.

4. Выбрать тип элемента интерфейса задающего параметр из списка **Type**:

- **Edit** – Редактируемое поле ввода.
- **Checkbox** – Флажок.
- **Popup** – Раскрывающийся список. В этом случае в графе **Popup Strings** (Элементы списка) необходимо ввести элементы списка, разделенные символом вертикальной черты. Например, выражение **alpha|beta|gamma** задаст список из трех элементов: **alpha**,**beta** и **gamma**.

5. Установить параметр **Evaluate** – Вычисляемый. Выбирается, если параметр должен иметь числовое значение. В данное поле можно будет ввести выражение в соответствии с правилами языка **MATLAB**. Формат **Evaluate** позволяет также использовать числовую форму значения переменной в том случае, если тип элемента интерфейса выбран в виде флажка или раскрывающегося списка. Так, например, для раскрывающегося списка **alpha|beta|gamma** значение связанной со списком переменной будет равно **1**, если в списке выбрано **alpha**, **2** – если в списке выбрано **beta**, и **3** – если в списке выбрано **gamma**. Для элемента интерфейса **Checkbox** вычисляемые значения будут равны **1** (при установленном флажке) и **0** (при снятом флажке).

6. Ввести на вкладке **Initialization** команды инициализации в графе **Initialization commands**. Команды инициализации представляют собой обычные команды на языке **MATLAB** и могут включать операторы и **m**-функции. Такие команды задают переменные, которые будут находиться в рабочей области маскированной подсистемы. Эти переменные доступны внутри подсистемы и могут быть использованы в качестве параметров блоков входящих в состав подсистемы, а также для создания пиктограммы подсистемы. Команды инициализации выполняются в следующих случаях:

- При открытии окна модели.
- При запуске модели на выполнение.

- При выполнении команды **Edit/Update diagram**.
- При вращении блока маскированной подсистемы (в этом случае команды инициализации обеспечивают перерисовку пиктограммы).
- При автоматическом изменении пиктограммы, зависящей от параметров блока.

Создание пиктограммы подсистемы

Пиктограмма подсистемы создается с помощью вкладки **Icon** (Пиктограмма) редактора маски.

Вкладка содержит следующие элементы:

1. **Drawing commands –** Область ввода команд рисования. Команды рисования являются выражениями допустимыми в языке **MATLAB**.

2. **Frame –** Список позволяющий выбрать способ отображения рамки пиктограммы:

- **Visible –** Рамка видна.
- **Invisible –** Рамка не видна.

3. **Transparency -** Список позволяющий установить прозрачность пиктограммы:

- **Opaque –** Пиктограмма не прозрачна.
- **Transparent–** Пиктограмма прозрачна.

4. **Rotation -** Список позволяющий задать возможность вращения пиктограммы:

- **Fixed –** Положение пиктограммы фиксировано.
- **Rotates –** Пиктограмма может вращаться вместе с блоком.

5. **Units –** Список, задающий условия масштабирования пиктограммы.

- **Autoscale –** Автоматическое масштабирование. Рисунок занимает максимально возможную площадь внутри пиктограммы.
- **Normalized –** Нормализованное масштабирование. Координаты левого нижнего угла пиктограммы **(0,0)**, координаты правого верхнего угла **(1,1)**.
- **Pixel –** Координаты рисунка задаются в пикселах.

Команды вывода текста

Для вывода текста могут использоваться следующие команды:

1. **disp('text')** или **disp(variablename) –** Вывод текста **'text'** или значения символьной переменной **variablename** в центре пиктограммы.

2. **text(x, y, 'text')** или **text(x, y, variablename) –** Вывод текста **'text'** или значения символьной переменной **variablename** начиная с позиции, заданной координатами **x** и **y**.

3. **text(x, y, 'text', 'horizontalAlignment', halign,'verticalAlignment', valign)-**Вывод текста **'text'** в позиции заданной координатами **x** и **y** и с указанием способов выравнивания относительно этой позиции по вертикали или горизонтали. Параметр **halign** может принимать значения: **'left', 'right'** или **'center'**. Параметр **valign** может принимать значения: **'base'**, **'bottom'** или **'middle'**.

4. **fprintf('text')** или **fprintf('format', variablename) –** Форматированный вывод (по правилам языка **C**) текста **'text'** или значения символьной переменной **variablename** в центре пиктограммы.

5. **port_label(port_type, port_number, label) –** Вывод на пиктограмме метки порта. Например, выражение **port_label('input', 1, 'a')** выводит на пиктограмме метку **a** первого входного порта.

Для вывода текста в несколько строк допускается использование сочетания символов **\n** для перехода на новую строку.

Команды построения графиков

Для построения графиков на пиктограмме могут использоваться следующие команды:

1. **plot(Y) –** В том случае, если **Y** является вектором, то строится график по оси абсцисс которого откладывается значение индекса элемента, а по оси ординат значение самого элемента. В том случае если **Y** является матрицей – строятся линии для каждого столбца. По оси абсцисс в этом случае также откладывается значение индекса элемента.

2. **plot(X1,Y1,X2,Y2,...) –** Строится графики вида **Y1(X1)**, **Y2(X2)** и т.д.

Команды отображения рисунка из графического файла

Для отображения на пиктограмме рисунка из графического файла используются следующие команды:

1. **image(imread('filename'))** – Отображение рисунка из файла с полным именем **filename**. Для правильной работы этой команды необходимо поместить рисунок в туже папку, где находится файл модели, и сделать эту папку рабочей. Допускается также совместно с именем файла указывать его полный путь.

2. **image(a, [x, y, w, h]) –** Отображение рисунка содержащегося в переменной **a**. Ширина и высота рисунка задаются параметрами **w** и **h**, соответственно. Левый нижний угол рисунка расположен в точке с координатами **x**,**y**. Считывание рисунка из файла может быть выполнено командой **a = imread('filename'))**.

3. **image(a, [x, y, w, h], rotation) –** Команда аналогичная предыдущей, но позволяющая задавать поведение рисунка при вращении пиктограммы. Значение параметра **rotation** равное **'on'** позволяет поворачивать рисунок вместе с пиктограммой подсистемы.

4. **patch(x, y) –** Отображение закрашенного многоугольника, координаты которого заданы векторами **x** и **y**. Цвет рисунка – черный.

5. **patch(x, y, [r g b]) -** Команда аналогичная предыдущей, но позволяющая задавать цвет рисунка. Параметры **r**,**g** и **b** задают соотношение красного, зеленого и синего цветов в рисунке. Значение параметров должно находиться в пределах от **0** до **1**.

Использование редактора пиктограмм iconedit

Для создания пиктограмм можно также использовать редактор пиктограмм **iconedit**. Для его вызова используется команда: **iconedit('modelname','Subsystem')**, где **modelname –** имя файла модели (без расширения), **Subsystem –** имя подсистемы, для которой будет создаваться пиктограмма.

Пиктограмма создается по точкам, расположение которых указывается с помощью мыши. Между собой точки соединяются прямыми линиями. Для

того, чтобы начать новую линию необходимо нажать клавишу **n** на клавиатуре. Для отмены создания последней точки используется клавиша **d**. Выход из редактора с автоматическим обновлением пиктограммы осуществляется клавишей **q**. По завершении работы с редактором необходимо также закрыть его окно рисования. Кроме обновления пиктограммы завершение работы с редактором пиктограмм сопровождается выводом в командной строке **MATLAB** графической команды, обеспечивающей построение пиктограммы.

2.12. Справочные подсистемы

Для создания описания и справки маскированной подсистемы служит вкладка **Documentation** (Документация). Вкладка **Documentation** содержит три графы: **Mask Type**, **Mask description** (Описание маски) и **Mask Help** (Справка по маске)

Текст, введенный в графу **Mask description**, отображается в верхней части окна диалога и предназначен для краткого описания блока. В графу **Mask Help** вносятся команды обеспечивающие загрузку файлов справки, созданных пользователем, в справочную систему при нажатии клавиши **Help** в окне параметров. Эти команды описаны в документации по **Simulink**. Наиболее удобным форматом файла справки является **htm** (**html**) – формат. Вызов справочного **htm**-файла осуществляется командой вида:

web(['file:///' which('1.asp')]); , где **helpfile.htm –** имя файла справки.

Для правильной работы справочной системы необходимо, чтобы файл справки находился в той же папке, что и файл модели, и данная папка являлась рабочей. Допускается также вместе с именем файла указывать его полный путь.

3. Пример разработки имитационной модели

Задание. Используя метод Монте-Карло, оценить площадь круга радиусом R, центр которого находится в точке M с координатами (A,B). Количество имитаций – N, количество точек, распределенных равномерно внутри квадрата, в который вписан круг на каждой имитации – n. Вектор результатов имитационного исследования вывести в файл и рабочую область Matlab. Для

расчета площади круга вычислить среднее значение из элементов полученного вектора, а также дисперсию для оценки качества имитационного моделирования.

Решение.

1. Установить блоки из раздела **Sources**, задающие начальные условия моделирования:

- **Clock** – для отсчета количества имитаций.
- **Constant** – N – для задания количества имитаций.
- **Constant** – R – для установления радиуса окружности.
- **Constant** – M – для установления координат центра окружности. Центр окружности задается в виде вектора: [A,B]
- **Constant** – n – для задания количества точек распределенных внутри квадрата.

2. Установить блок **For Iterator Subsystem**. Данный блок является подсистемой, которая позволяет выполнить определенные действия заданной число раз. В данном примере таким числом является параметр **n.** Двойной щелчок по данному блоку раскрывает его для редактирования.

3. В подсистеме **For Iterator Subsystem** необходимо разместить три входных параметра In1 из раздела Sources и один выходной параметр Out1 из раздела Sinks. После этого вернутся в главное окно задачи и подать сигналы: с блока **Constant** – R – на первый вход подсистемы **For Iterator Subsystem**, с блока **Constant** – M – на второй вход, с блока **Constant** – n – на третий вход.

4. В окне подсистемы **For Iterator Subsystem** соединить блок **In3** с блоком **For Iterator**.

5. Для генерации точек распределенных случайным образом внутри квадрата, использовать блок **MATLAB Fcn** (Раздел **User-Defined Functions**), в котором в параметр **MATLAB Function** необходимо записать имя функции – генератора координат точки – **RandXY(u(1),u(2),u(3)).** Аргументами данной функции являются соответственно – радиус, абсцисса и ордината центра окружности. Для правильной работы данной функции, необходимо, чтобы все

входные параметры были объединены в один вектор. Для этого служит блок **Mux** из раздела **Signal Routing.** Соединить блок **In1** с первым входом блока **Mux**, а **In2** – со вторым. Блок **Mux**, в свою очередь, должен быть соединен с блоком **MATLAB Fcn**. Результатом работы блока **MATLAB Fcn** будет являться вектор, содержащий случайные абсциссу и ординату точки.

Для создания функции **RandXY** выбрать команду **File – New – M file**. В появившемся окне ввести:

```
function Result=RandXY(r,A,B)
x=rand(1,1)*((A+r)-(A-r))+(A-r);
y=rand(1,1)*((B+r)-(B-r))+(B-r);
Result(1)=x;
Result(2)=y;
```

Функцию сохранить в файл с тем же именем – **RandXY**.

Для проверки работоспособности функции введите в командной строке окна MATLAB имя функции с конкретными параметрами. Например, **RandXY(5,1,2).** Результатом будет являться пара случайных координат точки внутри квадрата.

6. Для определения факта попадания точки в круг необходимо разместить два блока **Fcn** (Раздел **User-Defined Functions**). В первом – вычислить квадрат радиуса круга, предварительно соединив блок с блоком **In1**. А во втором значение выражения **(u[1]-u[3])^2+(u[2]-u[4])^2**, соединив блок с блоком новым **Mux**, который объединяет вектор, содержащий случайные абсциссу и ординату точки (выходной параметр блока **MATLAB Fcn**) и вектор координат центра окружности.

7. Подать выходные сигналы с предыдущих блоков на блок **Relational Operator** (раздел **Logic and Bit Operations**). Установить параметр **Relational Operator** – “>” – если первый входной сигнал – квадрат радиуса, и “<=” – в противном случае.

8. Выходной сигнал предыдущего блока, подать на блок **Sum** и организовать накопление количества точек, попавших в круг при помощи блока **Memory**.

9. Рассчитать площадь квадрата, используя блок **Fcn** по формуле **(2*u)^2**, соединив его с блоком **In1**.

10. Разместить блок **Divide** (раздел **Math Operations**). В параметре **Number of inputs** указать”**/”. Данная запись означает, что в данном блоке будут перемножены первый и второй входные сигналы, а результат будет поделен на третий сигнал, что соответствует формуле для нахождения площади круга: $S = \frac{Q \cdot Sk}{N}$, где Q – количество точек, попавших в круг (результат работы блоков **Sum** и **Memory**), Sk – площадь квадрата.

11. Блок **Divide** соединить с блоком **Out1**. В результате работы подсистемы на блок **Out1** будет подаваться площадь круга на данном шаге имитационного моделирования.

12. Для правильной работы модели необходимо для каждого блока указать параметр **Sample Time** равный -1, а для блока **Memory** установить флажок **Inherit sample time**.

13. Результатом работы подсистемы **For Iterator Subsystem** будет являться вектор значений площадей круга. Размерность данного вектора будет равна количеству имитаций.

14. В главном окне задачи организовать накопление суммы площадей круга при каждой имитации, используя блоки **Sum** и **Memory**. Это необходимо для расчета среднего значения площади круга.

15. Установить блок **Mux**. Подать на первый вход блока значение блока **Constant** – N, а на второй вход – сумму площадей круга.

16. Установить блок **Fcn**, в котором будет рассчитываться среднее значение площади. Соединить этот с блок блоком **Mux** (см. п. 15). В данном блоке ввести формулу **u(2)/u(1).** Вывести результат работы блока в блок

Display (раздел **Sinks**). Полученное значение будет являться средней оценкой площади круга.

17. Для вывода полученного вектора площадей круга в файл установить блок **To File** (раздел **Sinks**), а для вывода в рабочую область – блок **To Workspace** (раздел **Sinks**). Для блока **To Workspace** установить параметр Save format – Array, а в параметр Variable name ввести имя переменной, например – Y. Соединить выход блока **For Iterator Subsystem** с входами данных блоков. Значения полученного вектора можно проконтролировать, введя в командной строке окна MATLAB значение – Y и нажав клавишу Enter. Для расчета среднего значения вектора ввести – **mean(Y),** а для расчета дисперсии - **var(Y).**

18. Для расчета дисперсии непосредственно в **Simulink**, необходимо открыть библиотеку статистических блоков. Для этого в окне **Simulink Library Browser** необходимо выполнить команду **File – Open** и указать файл **C:\MATLAB701\toolbox\dspblks\dspblks\dspstat3.mdl**. Переместить блок Variance в окно задачи и соединить его с блоком **For Iterator Subsystem** . Установить флажок **Running Variance**. Результат вывести в блок **Display**.

19. Для остановки имитационной модели при достижении числа имитаций, указанных в блоке **Constant** – N необходимо подать сигналы с блоков **Constant** – N и **Clock** на блок **Relational Operator** (раздел **Logic and Bit Operations**). Установить параметр **Relational Operator** – "==". Результатом работы данного блока будет 0, если входные сигналы не равны и 1 в обратном случае. Соединить данный блок с блоком **Stop Simulation**.

20. Вывести текущий номер имитации, соединив блок **Clock** с блоком **Display**.

4. Задания к лабораторным работам

Задание 1. Время между последовательными прибытиями покупателей в магазине равномерно распределяется в интервале от 1 до 20 мин. Для 50% покупателей время обслуживания составляет 8 мин, в то время как для остальных 50% это время составляет 14 мин. Предложите подходящий для

этого случая генератор случайных чисел, отображающих времена прибытия покупателей, и другой такой генератор - для отображения времени обслуживания. Имитируйте 4 ч работы. Определите суммарное время ожидания покупателей и время простоя системы обслуживания.

Задание 2. Т. А. Шанс живет в Лас-Вегасе. Ежедневно перед работой он заходит в соседний ресторан, съедает свой завтрак и потом в ожидании автобуса, который доставляет его к месту работы, играет в «тройку». В этой игре игрок подбрасывает многократно монету до тех пор, пока разность между числом выпавших «гербов» и «решеток» не станет равна трем. За каждый бросок монеты игрок платит 1 долл., но при удачном исходе кона игры он получает 8 долл. Каждое утро Шанс откладывает 10 долл. для этой игры и играет в нее до тех пор, пока либо не проиграет все деньги, отложенные им в этот день на игру, либо не завершит одну партию. Определите с помощью имитационной модели результат игры Шанса в течение недели (5 рабочих дней), т. е. будет ли он в выигрыше?

Задача 3. Даны случайные некоррелированные переменные A, B и C. Переменная A имеет нормальное распределение с μ=100 и σ=20. Переменная B также распределена нормально с μ=20 и σ=5. Распределение переменной C задано в следующем виде:

Значение C	10	20	30	40
Вероятность	0,10	0,25	0,50	0,15

Применяя метод Монте-Карло, оцените среднее значение новой переменной Z, определяемой как Z=(A+B)/C, пользуясь выборкой из 10 значений.

Задача 4. Задача о «пьяном прохожем». Пусть вероятности перемещения от любого заданного перекрестка в различных направлениях имеют следующие значения: 50% для перемещения прямо, 20% для перемещения направо или налево и 10% для движения назад. Проведите моделирование этой ситуации и, использовав 10 попыток, вычислите вероятность того, что прохожий, пройдя

пять кварталов, закончит свой путь не далее чем в двух кварталах от начальной точки.

Задача 5. Потребность в некоторых деталях следует распределению Пуассона:

Ежедневная потребность	Вероятность	Число дней до поставки	Вероятность
0	0,30	2	0,10
1	0,36	3	0,50
2	0,22	4	0,30
3.	0,09	5	0,10
4	0,03		

Запас деталей проверяется в конце каждой пятидневной рабочей недели и если он оказывается равным шести штукам или менее, размещается заказ на десять деталей. Распределение времени выполнения заказов задается правой частью таблицы (время исчисляется в рабочих днях до поставки). Начальные условия моделирования: рабочая неделя только начинается, имеется в наличии десять деталей, заказов на новые детали нет. Промоделируйте пять рабочих недель. Постройте график уровня запаса к концу каждого рабочего дня и подсчитайте число дней, когда имеется нехватка деталей.

Задача 6. Начальник пожарной охраны обнаружил, что число пожаров за сутки следует распределению Пуассона со средним значением четыре пожара в сутки. Изучив данные по прежним пожарам, он нашел, что в 75% случаев для тушения потребовалась только одна пожарная машина, а время, необходимое для ликвидации пожара, имеет нормальное распределение с μ=3 ч и σ=0,5 ч. В остальных 25% случаев нужны были две пожарные машины, а время для ликвидации этих пожаров распределялось нормально с μ=4 ч и σ==1 ч. Предполагая, что необходимые пожарные машины всегда имеются в наличии, определите, сколько часов в среднем они бывают нужны каждые сутки. При моделировании возьмите объем выборки равным 10 суткам.

Число пожаров в сутки	Вероятность
0	0,02
1	0,07
2	0,15
3	0,20
4	0,20
5	0,16
6	0,10
7	0,06
8	0,03
9	0,01

Задача 7. Грузовики прибывают на разгрузочную станцию в случайные моменты времени. Анализ прежних записей показал, что темп прибытия грузовиков подчиняется распределению Пуассона со средним значением 3 грузовика в сутки.

Число прибывающих грузовиков	Вероятность
0	0,05
1	0,15
2	0,22
3	0,22
4	0,17
5	0,11
6	0,05
7	0,03

Вес груза, прибывающего с каждым грузовиком, является главным фактором, определяющим время разгрузки. Прежние записи показывают, что веса грузов распределены нормально со средним значением 12 т и среднеквадратическим отклонением 2 т. Скорость разгрузки, т. е. вес, который может обработать бригада грузчиков за 1 ч, также переменна, и зависит от вида груза. Вероятность прибытия груза каждого вида и соответствующие скорости разгрузки указаны в таблице. Бригада состоит из 3 человек - водителя автопогрузчика (часовая ставка 4 долл.) и двух рабочих (часовая ставка 2,50 долл.). Фирма следует правилу - разгружать все грузовики не позже, чем через сутки после прибытия. Все грузовики, прибывшие накануне, должны быть разгружены независимо от того, какую сумму сверхурочных придется

выплатить. Коллективный договор, заключенный с профсоюзом грузчиков, требует полуторной оплаты за каждый час, проработанный сверх 8-часового рабочего дня. Взяв объем выборки равным 10 суткам, определите, сколько бригад следует нанять, чтобы свести к минимуму общую стоимость разгрузки.

Вид груза	Вероятность	Скорость разгрузки, т/ч
A	0,40 \	3,2
B	0,35	2,8
C	0,25	2

5. Литература

1. Гультяев А. Визуальное моделирование в среде MATLAB: учебный курс – СПб: Питер, 2000. – 432 с.: ил.

2. Ануфриев И.Е. Самоучитель MATLAB 5.3/6.x. – СПб.: БХВ-Петербург, 2002. – 736 с.: ил.

3. Имитационное моделирование экономических процессов: Методические указания к выполнению лабораторных работ для студентов экономических специальностей./ Составители: Макарова И.В., Лысанов Д.М., Фрикк В.С. Набережные Челны: Изд-во КамПИ, 2005, 54 с.

Махмутов И.И., Шарипов Р.Ш.

Визуальное моделирование экономических процессов в среде MATLAB
Часть 2. Разработка имитационных моделей средствами пакета SIMULINK

Учебно-методическое пособие

Подписано в печать 22.04.2019
Формат 60x84/16. Печать ризографическая.
Бумага офсетная. Гарнитура «Times New Roman».
Усл.п.л. 2.75 Уч.-изд. л. 2.64
Тираж 100 экз. Заказ № 1245

Отпечатано в Издательско-полиграфическом центре
Набережночелнинского института
Казанского (Приволжского) федерального университета

423810, г. Набережные Челны, Новый город, пр.Мира, 68/19
тел./факс (8552) 39-65-99 e-mail: ic-nchi-kpfu@mail.ru