ЛЕКЦИЯ № 7 · Проект MATLAB IoT

Переходы по логическому сравнению величин. Целочисленное и вещественное сравнение для разветвляющихся алгоритмов



Бельгибаев Бауржан Абдрахимович
Д.т.н., доцент
Кафедра «Искусственный интеллект и big data»
ОП 6B07113 «Интеллектуальные системы управления»

Цель лекции

Изучить логические операции, правила сравнения целочисленных и вещественных величин, конструкции ветвления MATLAB и их использование для контроля технологических параметров и принятия решений в IoT-системах. Научиться создавать устойчивые к ошибкам измерений алгоритмы.

План лекции

1. Логические условия в инженерных алгоритмах
2. Числовые и логические типы данных MATLAB
3. Операторы сравнения и приоритет вычислений
4. Целочисленные сравнения и граничные значения
5. Особенности сравнения вещественных чисел
6. Операторы AND, OR, NOT и короткое замыкание
7. Конструкция if–elseif–else
8. Конструкция switch–case–otherwise
9. Векторные логические маски, any и all
10. Пороговый контроль и гистерезис
11. Обработка некорректных измерений и отказов
12. Сквозной пример мониторинга температуры и влажности

1 Логические условия в инженерных алгоритмах

Разветвляющийся алгоритм выбирает один из нескольких путей вычисления на основании логического условия. В системах автоматизации это необходимо для определения режима оборудования, обнаружения выхода параметров за пределы, выбора управляющего воздействия и формирования диагностических сообщений.

Простейший пример: если измеренная температура T превышает порог Tmax, программа выводит предупреждение; иначе сообщает о нормальном состоянии. Важно различать вычислительное решение и действие исполнительного механизма: лабораторная программа может моделировать предупреждение, но физическое отключение требует отдельного безопасного контура.

Логическое условие принимает значение true (истина) или false (ложь). В MATLAB результатом операции сравнения обычно является значение типа logical. Условия могут быть простыми, например T > 80, или составными, например T > 80 && pressure > 2.

Разветвления позволяют формализовать технологические правила, но сами правила должны быть обоснованы нормативами, характеристиками датчиков и условиями эксплуатации. Численный порог в учебном примере не является универсальной уставкой безопасности.

1

Логические условия в инженерных алгоритмах (продолжение)

При проектировании ветвления необходимо определить поведение для всех случаев: ниже порога, точно на пороге, выше порога, недостоверное измерение и отсутствие данных. Неописанный случай часто становится причиной программной ошибки.

2 Числовые и логические типы данных

В MATLAB обычные числовые литералы по умолчанию имеют тип double. Целочисленные типы включают int8, uint8, int16, uint16, int32, uint32, int64 и uint64. Логические значения создаются с помощью true и false, а также операций сравнения.

Целочисленные типы имеют ограниченные диапазоны. Например, uint8 хранит значения от 0 до 255. При преобразованиях и арифметике целочисленных типов нужно учитывать правила округления и насыщения, а также совместимость операндов.

Тип данных важен при взаимодействии с аппаратными интерфейсами: цифровой код АЦП может быть целым числом, а пересчитанная температура — вещественным. Сравнение сырого кода с температурным порогом без предварительного преобразования единиц математически некорректно.

Для определения типа используют class(x), для проверки целочисленности типа — isinteger(x), а для проверки логического типа — islogical(x). При необходимости число преобразуют в double(x), но это не восстанавливает информацию, утраченную при предшествующем округлении.

В MATLAB выражение x = 5 присваивает значение, тогда как x == 5 проверяет равенство. Путаница между присваиванием и сравнением — типичная ошибка начинающих.

2 Числовые и логические типы данных (продолжение)

```
raw = uint16(2048);
voltage = double(raw)*3.3/4095;
isHigh = voltage > 1.5;
disp(class(raw));
disp(class(voltage));
disp(class(isHigh));
```

3 Операторы сравнения и приоритет

В MATLAB применяются операторы < (меньше), <= (меньше либо равно), > (больше), >= (больше либо равно), == (равно) и ~= (не равно). Для скаляров результат является одним логическим значением; для совместимых массивов сравнение выполняется поэлементно.

Оператор >= включает граничное значение, а > исключает его. Например, при T=80 выражение T>80 ложно, а T>=80 истинно. При выборе оператора следует точно понимать формулировку технологического требования.

Арифметические операции обычно выполняются перед сравнением, однако сложные условия желательно группировать скобками. Запись (T >= Tmin) && (T <= Tmax) яснее и надёжнее, чем выражение без явного выделения условий.

Математическая запись 20 < T < 30 не должна переноситься в MATLAB буквально: выражение вычисляется последовательно и не означает проверку принадлежности интервалу. Правильная форма — (T > 20) && (T < 30).

Для массива температур условие T>80 возвращает массив логических значений. Такой результат не следует напрямую передавать в оператор &&, который предназначен для скалярных условий.

3 Операторы сравнения и приоритет (продолжение)

```
T = 80;
disp(T > 80);       % false
disp(T >= 80);      % true
inRange = (T >= 20) && (T <= 90);
disp(inRange);
samples = [75 81 79 84];
flags = samples > 80;
disp(flags);
```

4 Целочисленные сравнения

Целочисленное сравнение широко применяется для проверки цифровых кодов, счётчиков, идентификаторов, состояний протокола и номеров пакетов. Если два значения представлены одним и тем же целочисленным типом и не подвергались потере точности, проверка на точное равенство часто уместна.

Например, значение status=uint8(2) может означать состояние «предупреждение» в заранее определённом протоколе. Условие status==uint8(2) является точной проверкой кода. При этом значения кодов должны быть согласованы с документацией устройства.

При работе с 64-битными целыми числами важно не выполнять ненужное преобразование в double: числа больше 2^53 не все представимы точно в double. Из-за этого различные целые значения могут превратиться в одинаковые вещественные числа.

Пограничные значения проверяются отдельно: минимальный и максимальный допустимый код, отрицательные значения для знаковых типов и переполнение счётчика. В алгоритмах передачи данных полезно предусмотреть циклическую нумерацию и корректное обнаружение пропусков.

4 Целочисленные сравнения (продолжение)

Целочисленная точность относится к сохранённому коду, но не гарантирует физическую точность датчика. Например, код АЦП может быть целым и точным как число, хотя преобразованное напряжение содержит измерительную погрешность.

```
status = uint8(2);
if status == uint8(0)
    disp('Ожидание');
elseif status == uint8(1)
    disp('Работа');
elseif status == uint8(2)
    disp('Предупреждение');
else
    disp('Неизвестный код');
end
```

5 Сравнение вещественных чисел и численная точность

Вещественные числа в MATLAB обычно представлены двоичным форматом IEEE 754 двойной точности. Многие десятичные дроби, например 0,1, не имеют точного конечного двоичного представления. Поэтому результаты математически эквивалентных вычислений могут не совпадать побитно.

Проверка a==b для вычисленных вещественных величин часто ненадёжна. Например, значение 0.1+0.2 может не быть точно равным 0.3. Вместо точного равенства применяют сравнение с допуском: abs(a-b) <= tol.

Абсолютный допуск подходит, когда масштаб величины известен. Для значений различного порядка часто используют комбинированный критерий |a-b| <= atol + rtol*max(|a|,|b|), где atol — абсолютный, rtol — относительный допуск. Выбор допусков должен соответствовать точности измерений и требованиям задачи.

Численная погрешность вычисления и метрологическая погрешность датчика — разные явления. Если датчик имеет неопределённость ±0,5 °C, выбор машинного допуска 1e-12 не делает вывод о температуре физически точнее.

5 Сравнение вещественных чисел и численная точность (продолжение)

Для NaN обычные сравнения, включая NaN==NaN, возвращают false. Поэтому перед логическими проверками измерения следует проверять с помощью isfinite или isnan. Для значений Inf необходимо определить допустимость их появления в модели.

```
a = 0.1 + 0.2;
b = 0.3;
atol = 1e-12;
rtol = 1e-9;
isClose = abs(a-b) <= atol + rtol*max(abs(a),abs(b));
fprintf('Точное равенство: %d\n',a==b);
fprintf('Близость с допуском: %d\n',isClose);
```

6 Логические операции AND, OR, NOT

Логическая конъюнкция AND истинна, только если оба условия истинны; дизъюнкция OR истинна, если истинно хотя бы одно; отрицание NOT меняет true на false и наоборот. В MATLAB для массивов применяются &, | и ~.

Операторы && и || используются для скалярных условий и обладают коротким замыканием: если результат уже определён по первому операнду, второй может не вычисляться. Это полезно для защиты от недопустимого обращения к данным.

Например, условие ~isempty(T) && isfinite(T(1)) сначала проверяет, что массив не пуст, и только затем обращается к первому элементу. Если использовать поэлементный &, второй операнд также может вычисляться, что способно вызвать ошибку индексации.

При обработке массивов применяют логические маски: valid=isfinite(T) & (T>=0) & (T<=100). Результат содержит true для допустимых отсчётов. Функция sum(valid) подсчитывает количество истинных элементов.

При проектировании сложного условия полезно записывать его смысл словами и проверять несколько комбинаций входных значений. Это предотвращает ошибки при смешении AND и OR.

6 Логические операции AND, OR, NOT (продолжение)

```
T = [24.1 26.2 NaN 105 28.3];
valid = isfinite(T) & (T >= 0) & (T <= 100);
goodT = T(valid);
fprintf('Допустимых: %d\n',sum(valid));
disp(goodT);
```

7 Конструкция if–elseif–else

Условный оператор if выполняет блок команд, если логическое условие истинно. Дополнительные ветви elseif проверяются последовательно только в случае, если предыдущие условия ложны. Ветвь else выполняется, когда ни одно из предыдущих условий не оказалось истинным.

Порядок ветвей имеет принципиальное значение. Например, если сначала проверить T>60, а затем T>80, то температура 85 °C попадёт в первую ветвь и до второй проверка не дойдёт. Поэтому более строгие условия обычно размещают раньше.

При диагностике можно выделить состояния «норма», «предупреждение», «аварийное превышение» и «ошибка датчика». Проверка достоверности должна предшествовать сравнению с технологическими порогами.

Конструкция if работает со скалярным логическим результатом. Для проверки всего массива используют any(condition) или all(condition), в зависимости от задачи. Применение any означает «существует хотя бы один», а all — «все элементы удовлетворяют».

Для безопасного программирования желательно предусмотреть ветвь else на случай неожиданного значения. Она особенно важна при обработке кодов состояния устройства и сетевых сообщений.

7 Конструкция if–elseif–else (продолжение)

```
T = 84.5;
if ~isfinite(T)
    disp('Ошибка измерения');
elseif T >= 90
    disp('Критический уровень');
elseif T >= 80
    disp('Предупреждение');
else
    disp('Норма');
end
```

8 Конструкция switch–case–otherwise

Конструкция switch удобна, когда программа выбирает действие по одному значению или коду состояния. Она повышает читаемость по сравнению с длинной цепочкой elseif, особенно если варианты являются взаимоисключающими.

В MATLAB после switch expression следуют ветви case value, а в конце при необходимости указывается otherwise. Для кодов протокола это позволяет явно описать состояния устройства.

Например, код 0 может означать ожидание, 1 — измерение, 2 — передачу, 3 — ошибку. Значения кодов в учебном примере условны; для реального оборудования их необходимо брать из документации.

Конструкция switch не предназначена для проверки интервалов вида T>80. В таких случаях удобнее if–elseif–else. Выбор конструкции определяется смыслом условия, а не только предпочтением программиста.

Ветвь otherwise помогает корректно обработать неизвестные коды и не допустить молчаливого пропуска ошибочного состояния.

8 Конструкция switch–case–otherwise (продолжение)

```
mode = uint8(2);
switch mode
    case uint8(0)
        disp('Ожидание');
    case uint8(1)
        disp('Сбор данных');
    case uint8(2)
        disp('Передача данных');
    otherwise
        disp('Неизвестный режим');
end
```

9 Логические маски, any и all

Логическая индексация — способ выбора элементов массива на основании условия. Например, T(T>80) возвращает только температуры выше 80 °C. Этот приём полезен для подсчёта превышений, удаления недостоверных измерений и анализа отдельных режимов.

Функция any(A) проверяет наличие хотя бы одного истинного элемента вдоль первого измерения, размер которого больше единицы; all(A) проверяет истинность всех элементов. Для вектора результат скалярный, но для матрицы по умолчанию может быть вектором по столбцам.

Чтобы проверить все элементы матрицы, удобно использовать any(A(:)) или all(A(:)). Явное преобразование в столбец с помощью A(:) делает намерение понятным и предотвращает ошибки.

При работе с матрицей нескольких датчиков можно формировать маску превышений для каждого канала, а затем подсчитывать количество превышений по столбцам командой sum(mask,1).

Логическая маска сохраняет связь с исходным индексом. Если удалять элементы массива, необходимо одновременно удалять соответствующие временные метки, иначе временной ряд станет некорректным.

9 Логические маски, any и all (продолжение)

```
M = [75 77 78; 82 79 80; 85 83 77; 76 81 79];
alarm = M > 80;
counts = sum(alarm,1);
hasAlarm = any(alarm(:));
allSafe = all(M(:) <= 90);
disp(counts);
fprintf('Есть превышение: %d\n',hasAlarm);
fprintf('Все <= 90: %d\n',allSafe);
```

10 Пороговый контроль и гистерезис

Простое пороговое правило может часто переключать состояние при колебаниях измерения около границы. Например, если порог 80 °C, последовательность 79,9; 80,1; 79,8; 80,2 вызывает чередование сообщений «норма» и «предупреждение». В реальной системе это может приводить к нежелательным частым переключениям.

Гистерезис вводит два порога: Ton для включения предупреждения и Toff для его отключения, причём Toff < Ton. Например, предупреждение включается при T>=80 °C, а снимается только при T<=78 °C. Между этими границами сохраняется прежнее состояние.

Такой алгоритм имеет память: результат зависит не только от текущего измерения, но и от предыдущего состояния. Поэтому его удобно реализовать циклом for с логической переменной alarm.

Гистерезис уменьшает дрожание переключений, но не заменяет фильтрацию, диагностику датчика и анализ динамики процесса. Пороги следует выбирать с учётом допустимого режима, точности измерения и риска задержки реакции.

Для реального аварийного отключения гистерезис и временные задержки должны проходить отдельную проверку безопасности. В данном примере формируется только программный индикатор.

10

Пороговый контроль и гистерезис (продолжение)

```
T = [77.0 79.0 80.2 79.7 78.9 77.8 79.5 80.4];
Ton = 80; Toff = 78;
alarm = false;
history = false(size(T));
for k = 1:numel(T)
    if ~alarm && T(k) >= Ton
        alarm = true;
    elseif alarm && T(k) <= Toff
        alarm = false;
    end
    history(k) = alarm;
end
disp(table(T(:),history(:), ...
    'VariableNames',{'Temperature_C','Alarm'}));
```

11 Недостоверные данные и отказоустойчивость

IoT-система получает данные из физических датчиков и по сетевым каналам. Возможны пропуски, NaN, Inf, значения за пределами диапазона, устаревшие сообщения и нарушение последовательности. Такие ситуации требуют отдельной ветви обработки.

При недостоверном измерении нельзя автоматически считать, что параметр находится в норме. Например, выражение NaN>80 возвращает false, что без предварительной проверки может ошибочно трактоваться как отсутствие превышения.

Проверка может включать isfinite(T), ограничение физического диапазона и контроль возраста сообщения. Для данных температуры 0–100 °C это лишь пример допустимого диапазона; в реальной задаче границы выбирают по технологическим условиям.

При отказе датчика алгоритм может формировать состояние «нет достоверных данных» и запрещать вычисление производных показателей. Для системы с физическим управлением переход в безопасное состояние должен обеспечиваться локальной защитой.

11

Недостоверные данные и отказоустойчивость (продолжение)

При тестировании полезно проверить не только обычные значения, но и ровно пороговые, NaN, Inf, отрицательные, резко меняющиеся и отсутствующие значения. Это позволяет выявить ошибки в логике до подключения оборудования.

```
T = [25 79 81 NaN Inf -10 83];
for k = 1:numel(T)
    if ~isfinite(T(k))
        fprintf('%d: недостоверное значение\n',k);
    elseif T(k) < 0 || T(k) > 100
        fprintf('%d: вне допустимого диапазона\n',k);
    elseif T(k) >= 80
        fprintf('%d: превышение порога\n',k);
    else
        fprintf('%d: норма\n',k);
    end
end
```

12

Сквозной пример: мониторинг температуры и влажности

Постановка задачи: имеются учебные измерения температуры и относительной влажности, поступающие с условного IoT-устройства. Необходимо классифицировать каждый отсчёт как «норма», «предупреждение» или «недостоверные данные», подсчитать число предупреждений и построить график температуры.

Для примера примем предупреждение при температуре не ниже 30 °C или относительной влажности не ниже 75 %. Допустимый диапазон входных значений температуры условно установлен от −20 до 100 °C, влажности — от 0 до 100 %. Эти пороги выбраны для учебного расчёта и не являются нормативными.

Алгоритм сначала проверяет, что оба значения конечны и входят в допустимые диапазоны. Если хотя бы одно недостоверно, формируется соответствующее состояние. Только после этого проверяются пороги. Такой порядок исключает трактовку NaN как нормального режима.

Для каждого отсчёта сохраняется строковое состояние, а также логический признак предупреждения. В конце вычисляется количество предупреждений и строится график с горизонтальной линией температурного порога.

12

Сквозной пример: мониторинг температуры и влажности (продолжение)

При переходе к реальным данным Raspberry Pi потребуется заменить учебные массивы импортом CSV или чтением API. Логика классификации останется применимой при условии, что единицы измерения и структура данных согласованы.

```
clear; clc; close all;
t = 0:10:70;
T = [25.0 26.2 29.5 30.3 29.7 NaN 31.1 28.9];
H = [60 63 70 72 77 74 76 65];
N = numel(t);
assert(numel(T)==N && numel(H)==N);
status = strings(1,N);
warning = false(1,N);
for k = 1:N
    valid = isfinite(T(k)) && isfinite(H(k)) ...
        && T(k)>=-20 && T(k)<=100 ...
        && H(k)>=0 && H(k)<=100;
    if ~valid
```

12

Сквозной пример: мониторинг температуры и влажности (продолжение)

```
        status(k) = "Недостоверно";
    elseif T(k)>=30 || H(k)>=75
        status(k) = "Предупреждение";
        warning(k) = true;
    else
        status(k) = "Норма";
    end
end
disp(table(t(:),T(:),H(:),status(:), ...
    'VariableNames',{'Time_s','Temp_C','Humidity_pct','Status'}));
fprintf('Предупреждений: %d\n',sum(warning));
plot(t,T,'-o','LineWidth',1.5); hold on;
yline(30,'--','Порог 30 °C');
grid on; xlabel('Время, с'); ylabel('Температура, °C');
title('Учебный мониторинг технологических параметров');
```

3 Заключение

В лекции рассмотрены логические типы данных, операции сравнения целых и вещественных величин, особенности вычислений с плавающей точкой, конструкции if–elseif–else и switch–case, а также логическая индексация массивов. Показано, как применять ветвления к контролю технологических параметров, гистерезису и обработке недостоверных показаний. При создании инженерного алгоритма важно проверять границы, учитывать точность измерений и явно описывать поведение системы при ошибках.

Контрольные вопросы (1/2)

1. Что такое разветвляющийся алгоритм?
2. Какой тип данных возвращает оператор сравнения?
3. Чем отличается присваивание = от сравнения ==?
4. Какие целочисленные типы доступны в MATLAB?
5. Почему преобразование int64 в double может приводить к потере точности?
6. Чем различаются > и >=?
7. Почему нельзя использовать запись 20<T<30?
8. Как выполняется поэлементное сравнение массива?
9. Почему точное равенство вещественных чисел часто ненадёжно?
10. Как выбрать абсолютный и относительный допуски?
11. Чем различаются численная и измерительная погрешности?
12. Что означают логические операции AND, OR и NOT?
13. Чем отличаются & и &&?
14. Что означает короткое замыкание?

Контрольные вопросы (2/2)

15. Как работает конструкция if–elseif–else?

16. Почему важен порядок ветвей elseif?

17. Когда удобно использовать switch–case?

18. Для чего предназначена ветвь otherwise?

19. Как использовать логическую маску для отбора данных?

20. Чем отличаются функции any и all?

21. Как проверить все элементы матрицы на выполнение условия?

22. Что такое гистерезис и зачем он нужен?

23. Почему NaN может приводить к ложному выводу об отсутствии аварии?

24. Как организовать проверку диапазона измерений?

25. Какие тестовые случаи необходимы для проверки алгоритма мониторинга?

Д Самостоятельные задания

1. Разработайте программу классификации температуры по трём уровням: ниже 20 °C, от 20 до 30 °C включительно, выше 30 °C.

2. Сравните два результата вычисления 0.1+0.2 и 0.3, реализуйте комбинированный критерий близости.

3. Для массива из 20 измерений определите число превышений порога и число недостоверных отсчётов.

4. Создайте switch–case для кодов состояния устройства 0, 1, 2, 3 и неизвестного кода.

5. Реализуйте гистерезис с порогами 75 и 72 °C и сравните с простым пороговым алгоритмом.

6. Разработайте программу мониторинга двух параметров с проверкой NaN, Inf и выходов за физический диапазон.

Рекомендуемая литература

• Силлабус дисциплины «Проект MATLAB IoT», тема лекции №7.

• MathWorks. MATLAB Documentation — https://www.mathworks.com/help/matlab/

• MathWorks. Relational Operations — https://www.mathworks.com/help/matlab/matlab_prog/array-comparison-with-relational-operators.html

• MathWorks. Conditional Statements — https://www.mathworks.com/help/matlab/control-flow.html