10. Контрол на потока (информация)

В предни раздели използвахме само една употреба на логическите променливи, избирайки елементи от матрица. Логическите променливи имат и втора употреба в програмата контрол на потока. Тази команда позволява различен код да бъде изпълнен в зависимост от това дали конкретни условия са на лице. Две структури за контрола са в наличност.

if …elseif…else u switch…case…otherwise.

10.1 If Elseif Else

if …elseif…else са по обикновените от двата. Тези блокове винаги започват с *if* твърдение, веднага последвано от s**calar** логически израз и трябва да бъдат ограничени с **end**. *elseif* и *else* са опции и могат винаги да бъдат репликирани използвайки вгнездени *if* твърдения и логически комплекси. Основната форма на *if …elseif…else* е

```
if logical_1
        Code to run if logical_1 true
elseif logical_2
        Code to run if logical_2 true and logical_1 false
elseif logical_3
        Code to run if logical_3 true and logical_j false, j < 3
...
...
else
        Code to run all logical's false
end
```

Обаче по простите форми са по често използвани:

```
if logical_1
        Code to run if logical_1 true
end
```

or

```
if logical_1
        Code to run if logical_1 true
else
        Code to run if logical_1 false
end
```

Забележка: Не забравяйте, че всички логични трябва да бъдат скаларни логични стойности.

Няколко прости примера:

```
>> x = 5;
>> if x<5
     x=x+1;
   else
     x=x-1;
   end
>> x
ans =
    4
```

и

```
>> x = 5;
>> if x<5
     x=x+1;
   elseif x>5
     x=x-1;
   else
     x=2*x;
   end
>> x
ans =
    10
```

Тези примери във всичките са използван прости логически изрази. Обаче,някои скаларни логически изрази такива като **(x<0 || x>1) && (y<0 || y>1)** или *isinf(x)* || *isnan(x)*, може би са *if …elseif…else* блокове.

10.2 Switch Case Otherwise

switch…case…otherwise са по напреднал контрол на потока и могат да бъдат изцяло репликирани, използвайки само *if …elseif…else* блокове на контрол на потока. Не се чувствайте задължени да използвате тези, ако не Вие удобно с тях. Основната структура на този блок е да намери някоя променлива, чиято стойност може да бъде използвана, за да избере парче (част) от кода и да го изпълни (променливата ***switch***). В заявка от стойността на тази променлива (нейния ***case***), конкретна част от кода ще бъде изпълнена. Ако никакви случаи не са свързани (**otherwise**) не изпълнен код от кода се изпълнява (**otherwise** може безопасно да бъде пропусната. В този случаи нищо не се върши, ако един от *cases* не са свързани (не съответстват). Обаче, с командата *at most* един блок е свързан. Свързването на ***case*** – командата заставя този кодов блок да изпълни, тогава програмата продължава на следващия ред след *switch…case…otherwise* блок. Основната форма на блока *switch…case…otherwise* е

```
switch variable

case value_1
        Code to run if variable = value_1 true

case value_2
        Code to run if variable = value_2 true

case value_3
        Code to run if variable = value_3 true

...

...

otherwise
        Code to run if variable ≠ value_j ∀j

end
```

Забележка: Има равновесие между *switch…case…otherwise* и *if …elseif…else* блоковете. Обаче, логическите изрази съдържат не равенства, логическите променливи трябва да бъдат създадени приоритетно за използване на *switch…case…otherwise* блока. Също ако имате C- програмираща среда, поведението на *switch…case…otherwise* е различно: единият случаи може да бъде свързан към един блок. След като първият е свързан, блокът е свален и програмата обобщава със следващия ред след блока.

Един прост пример със *switch…case…otherwise:*

```
x=5;
switch x
    case 4
        x=x+1;
    case 5
        x=2*x;
    case 6
        x=x-2;
    otherwise
        x=0;
end
 >> x
 ans =
    10
```

cases може да включва крайни стойности за *switch* променлива, използвайки обозначението **case** {$case_1$, $case_2$,…}. Например:

```
x=5; switch x
    case {4}
        x=x+1;
    case {1,2,5}
        x=2*x;
    otherwise
        x=0;
end
```

```
>> x
ans =
   10
```

11. Loops (Цикли)

Loops са най-полезната програмна структура намираща в Матлаб и може да прави много проблеми, особено когато се комбинира с блоковете на контрол на потока. Матлаб има *loops* от два типа, ***for … end*** и ***while … end***. **for loops** прави примка над предварително определения iterator и докато примките свършат, когато някои логически условия са на лице. Всичко за примките може да бъде изразено, като **while loops**, въпреки че противоположно е **not** истина. Те са почти еквивалент, когато *break* се използва, въпреки че кода е ненужно усложнен в тази ситуация. Две други команди, **return** и **break** са също полезни в кода на *loops*.

11.1 for

for *loops* започват с **for** *iterator = vector* и свършват с **end**.Основна структура на една *for loops* е:

```
for iterator=vector
      Code to run
end
```

iterator е променливо име*, loops* е повтаряща се през. Например, **i** е обикновен *iterator, vector* е вектор от данни. Той може да бъде съществуващ вектор или той може да бъде създаден в движение, използвайки **linspace** или **a:b:c** синтаксис (т.е. 1:10). Един тънък аспект на loops в Матлаб, е че *iterator* (повтарача) може да съдържа, който и да е вектор данни, включително не цяло число and/or отрицателни стойност. Вземете под внимание тези три примера:

```
count=0;
for i=1:100
   count=count+i;
end

count=0;
for i=linspace(0,5,50)
   count=count+i;


 end

 count=0;
 x=linspace(-20,20,500);
 for i=x
    count=count+i;
 end
```

Първата *loops* ще се повтори *през i= 1,2,…,100.* Каква е крайната стойност на броенето? Вторите *loops* през стойностите, създадени от функцията *linspace*, която създава

50 еднакви точки между 0 и 5, включително. Последните *loops* през *x*, един вектор конструиран от повишаван в *linspace*. *Loops* могат също да се повтарят през намаляващи последователности (съответствия):

```
count=0;
x=-1*linspace(0,20,500);
for i=x
    count=count+i;
end
```

or vector with no order:

```
count=0;
x=[1 3 4 -9 -2 7 13 -1 0];
for i=x
    count=count+i;
end
```

Ключът за разбиране поведението на ***for*** *loops* , е че Матлаб винаги се повтаря през елементите на *vector* в реда, в който са представени (например, *vector(1), vector(2),….). Loops* могат също за бъдат вгнездени:

```
count=0; for i=1:10
    for j=1:10
        count=count+j;
    end
end
```

и могат да съдържат променливи на контрол на потока:

```
returns=randn(100,1); count=0; for i=1:length(returns)
    if returns(i)<0
        count=count+1;
    end
end
```

Една особено полезна примкова конструкция е да впримчи през **length** на вектор, като под волево всеки елемент да бъде модифициран по един всеки път.

```
trend=zeros(100,1);
for i=1:length(trend)
    trend(i)=i;
end
```

Накрая тези идеи могат да бъдат комбинирани за да създадат вгнездени разклонения с поточния контрол.

```
matrix=zeros(10,10);
for i=1:size(matrix,1)
    for j=1:size(matrix,2)
        if i<j
            matrix(i,j)=i+j;
        else
            matrix(i,j)=i-j;
        end
    end
end
```

Вие бихте дори могли да конструирате *loops* с вгнездени *loops*, които зависят от някое твърдение на поточния контрол.

```
matrix=zeros(10,10);
for i=1:size(matrix,1)
    if (i/2)==floor(i/2)
        for j=1:size(matrix,2)
            matrix(i,j)=i+j;
        end
    else
        for j=1:size(matrix,2)
            matrix(i,j)=i-j;
        end
    end
end
```

Забележка: Вие ***NOT*** трябва да модифицирате *iterato*r –та променлива вътре в предната *loops* (предходната). Променливата на *iterator* можа да създаде нежелани резултати. Например:

```
for i=1:100;
    i
    i=1;
    i
end
```

създава изход

```
...
...
 i =
    99
i =
     1
i =
   100
i =
     1
```

което може да доведе до непредсказуеми резултати, ако **i** е използван вътре в *loops*.

11.2 while (докато)

while *loops* са полезни, когато броя на необходимите повторения е неизвестен. *while loops* обикновено се използват, когато една *loops* би трябвало сама да спре, ако определено условие е налице, такова като промяната в някои параметър е малка. Основната структура на *while loops* е:

```
while logical
        Code to run
        Update to logical inputs
end
```

Две неща са особено важни, когато използуваме *while loops* :

1. *logical* би трябвало да е вярно, когато loops започва (или разклонението ще бъде игнорирано);
2. входовете към логическата променлива трябва да бъдат оставени временно вътре в *loops*. Ако те не са loops ще продължава завинаги (комбинацията от CTRL+C за да разрушиш грешната *loops*). Най-простите *while loops* се включват със заместванията за *for loops*:

```
count=0;
i=1;
while i<=10
    count=count+i;
    i=i+1;
end
```

което създава същите резултати като:

```
count=0;
for i=1:10
    count=count+i;
end
```

while *loops* би трябвало изобщо да се избягват, когато *for loops* ще се използват, обаче има ситуации, където не съществува еквивалент на for:

```
mu=1;
index=1;
while abs(mu)>.0001
    mu=(mu+randn)/index;
    index=index+1;
end
```

В блока по-горе, броят на изискваните повторения не е известен предварително и понеже **randn** е стандартно, нормално число, то може да отнеме много повторения, докато този критерий е налице. Която и да е ограничена ***for*** *loops* не може да бъде гарантирана за наличието на критерий.

11.3 break

break може да бъде използван за да прекъсне примка и може да прави поведението на ***for*** *loops* почти като на *while loop*:

```
for iterator=vector
      Code to run
      if logical
            break
      end
end
```

Единствената разлика между тази примка и стандартната *while loop* , е че *while loop* би могла потенциално да задейства повече повторения и *iterator*. Командата **break** може също да бъде използвана да прекрати *while loop* преди въвеждането на код вътре в нея. Вземете в предвид тази малко страна примка:

```
while 1
    x = randn;
    if x<0
        break
    end
    y = sqrt(x);
end
```

Използването на *while 1* ще създаде примка, ако е оставено само това ще върви недефинирано. Обаче командата **break** ще спре примката, ако някакво условие е налице. По-важно, е че командата **break** ще предпази кода след нея от това да бъде въведен, което е полезно, ако операциите **break** ще създадат грешки, ако условието не е вярно.

11.4 continue (продължение)

continue**,** когато се използва вътре в примката има ефектът за преместване на примката към следващо повторение и прескачане на някои оставащ код в тялото на примката. Нейното използване може да бъде избегнато, използвайки блокове *if …else*, но то може да направи кода по-подреден. Нейният ефект се вижда най-добре при блок на код:

```
for i=1:10
    if (i/2)==floor(i/2)
        continue
    end
    i
end
```

който създава изход

```
...
...
i =
     7
i =
     9
```

демонстриращ, че командата **continue** принуждава примката към следващо повторение, когато винаги ***i*** е даже *((i /2) = = floor(i / 2)* е логично вярно).