Loop Control −− for, while, continue, break

With loop control statements, you can repeatedly execute a block of code, looping back through the block while keeping track of each iteration with an incrementing index variable. Use the`for` statement to loop a specific number of times. The `while` statement is more suitable for basing the loop execution on how long a condition continues to be true or false. The`continue` and `break` statements give you more control on exiting the loop.

for

The `for` loop executes a statement or group of statements a predetermined number of times. Its syntax is

```
for index = start:increment:end
    statements
end
```

The default increment is`1`. You can specify any increment, including a negative one. For positive indices, execution terminates when the value of the index exceeds the *end* value; for negative increments, it terminates when the index is less than the end value.

For example, this loop executes five times.

```
for n = 2:6
    x(n) = 2 * x(n - 1);
end
```

You can nest multiple `for` loops.

```
for m = 1:5
    for n = 1:100
        A(m, n) = 1/(m + n - 1);
    end
end
```

> **Note** You can often speed up the execution of MATLAB code by replacing `for` and `while` loops with vectorized code. SeeVectorizing Loops for details.

Using Arrays as Indices. The index of a `for` loop can be an array. For example, consider an m−by−n array`A`. The statement

```
for k = A
    statements
end
```

sets `k` equal to the vector `A(:,i)`, where `i` is the iteration number of the loop. For the first loop iteration, `k` is equal to `A(:,1)`; for the second, `k` is equal to `A(:,2)`; and so on until `k` equals `A(:,n)`. That is, the loop iterates for a number of times equal to the number of columns in`A`. For each iteration, `k` is a vector containing one of the columns of `A`.

while

The while loop executes a statement or group of statements repeatedly as long as the controlling expression is true (1). Its syntax is

```
while expression
    statements
end
```

If the expression evaluates to a matrix, all its elements must be1 for execution to continue. To reduce a matrix to a scalar value, use theall and any functions.

For example, thiswhile loop finds the first integer n for which n! (n factorial) is a 100−digit number.

```
n = 1;
while prod(1:n) < 1e100
    n = n + 1;
end
```

Exit a while loop at any time using the break statement.

while Statements and Empty Arrays. A while condition that reduces to an empty array represents afalse condition. That is,

```
while A, S1, end
```

never executes statementS1 when A is an empty array.

continue

The continue statement passes control to the next iteration of the for or while loop in which it appears, skipping any remaining statements in the body of the loop. In nested loops, continue passes control to the next iteration of the for or while loop enclosing it.

The example below shows acontinue loop that counts the lines of code in the file, magic.m, skipping all blank lines and comments. A continue statement is used to advance to the next line inmagic.m without incrementing the count whenever a blank line or comment line is encountered.

```
fid = fopen('magic.m', 'r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line, '%', 1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines', count));
```

break

The break statement terminates the execution of a for loop or while loop. When a break statement is encountered, execution continues with the next statement outside of the loop. In nested loops, break exits from the innermost loop only.

The example below shows a while loop that reads the contents of the file fft.m into a MATLAB character array. A break statement is used to exit the while loop when the first empty line is encountered. The resulting character array contains the M−file help for the fft program.

```
fid = fopen(’fft.m’, ’r’);
s = ’’;
while ~feof(fid)
   line = fgetl(fid);
   if isempty(line)
      break
   end
   s = strvcat(s, line);
end
disp(s)
```

◀ Conditional Control −− if, switch   Error Control −− try, catch ▶