

## MATLAB Marina: Iteration, while loops advanced

### Student Learning Objectives

After completing this module, one should:

1. Be able solve problems using while loops.

### Terms

NA

### MATLAB Functions, Keywords, and Operators

break, continue, flag

### Approximating an Infinite Series

The Taylor series of a real valued (or complex valued) function that is infinitely differentiable at a value  $a$  is

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!} (x-a)^k$$

Where  $f^{(k)}(a)$  is the  $k$ th derivative of  $f(x)$  with respect to  $x$  evaluated at  $x = a$  and  $k!$  is the factorial of  $k$ . The Taylor series for the function  $f(x) = e^x$  at  $a = 0$  is

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

The infinite Taylor series can be approximated by using the  $n$ th Taylor polynomial  $P_n(x)$  where

$P_n(x) = \sum_{k=0}^n \frac{f^{(k)}(a)}{k!} (x-a)^k$ . For example, for  $n = 5$  (6 terms including  $k = 0$ ), the approximation

$$\text{is } e^x \approx P_5(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!}.$$

The value for  $n$  can be chosen as a set number of terms or based on an error bound, often how the small the terms being added to the sum have become. If the series is to be approximated using a set number of terms, array operations or a for loop can be used to determine the approximation. If, however, the series will be approximated using an error bound then a while loop will need to be used since the number of terms needed to meet the error bound is not known.

The program of Figure 1 approximates the function  $f(x) = e^x$  using an error bound corresponding to the magnitude of the term. If the term magnitude for the  $n+1$  term is sufficiently small, the approximation is complete using  $P_n(x)$  as the approximation.

---

```

% Maclaurin series for e^x (Taylor series about a = 0)
x = 2;
% term magnitude to stop approximation at
epsilon = 0.01;
% term for n = 0
P = 0.0;
n = 0;
Pn = x^n/factorial(n);
% add term to result and compute terms until desired accuracy
met
while(abs(Pn) >= epsilon)
    % add term to result
    P = P + Pn;
    % compute next term in series
    n = n + 1;
    Pn = x^n/factorial(n);
end

fprintf('Approximation of e^x for x = %0.1f is %0.2f.\n',x, P);

```

---

Figure 1. Program to Determine Approximation of  $f(x) = e^x$

The initial approximation is set to zero and the  $n = 0$  term is determined. The magnitude for the term for  $n = 0$  is compared to the error bound (0.01 here). If the term is larger than the error bound, then the following two operations are repeated until the term for a value of  $n$  is less than the error bound:

- New approximation is the old approximation plus the term (running sum of terms)
- $n$  is incremented and the new term for  $n$  is determined

When the term's magnitude for a particular  $n$  is less than the error bound, the approximation is complete and uses the terms from 0 to  $n-1$ .

### Loop break and continue

The break and continue statements give a programmer additional flexibility with loop control statements. The break statement ends the loop containing it and sends control to the next statement after the loop. For a nested loop, break causes the exit from the innermost loop. The continue command ends the current iteration of the loop sending control to the end of the loop body, i.e. skips the rest of the statements in the loop body and starts the next iteration. Break and continue are usually used as part of a conditional statement (if) nested within the loop. The break and continue statements can only be used with for and while loops.

Programs can usually be written without using break and continue statements and one should avoid using break and continue statements unless it makes the program easier to read. Using conditional structures to avoid performing operations or ensuring the logical expression of a while loop handles all the cases of terminating the loop is generally preferable to using break and continue.

The programs of Figures 2 and 3 illustrate the use of break and continue. The program of Figure 2 uses a break statement to exit the loop if a data value was identified as not valid. A for loop with a loop control array of 1:1:length(data) could be used in place of the while loop. The break statement in the program of Figure 2 could be eliminated if the test of dataGood was added to the loop control expression, k <= length(data) && dataGood.

---

```
data = [64, 36, 98, 23, 66, 61, NaN, 15, 39, 43];
dataGood = true;
k = 1;
while (k <= length(data))
    if isnan(data(k))
        dataGood = false;
        break;
    end
    k = k + 1;
end

if (dataGood)
    fprintf('Data is all valid\n');
else
    fprintf('Data element %d is not valid.\n',k);
end
```

---

Figure 2. Program Determining if all Data in Array is Valid

The program of Figure 3 sums the elements of the array data and omits any non-valid data (NaN, inf, -inf) from the sum. The while loop of Figure 4 performs the same operation but avoids the use of the continue statement and so is generally considered preferable.

---

```
% sum only numeric data
data = [64, 36, inf, 23, 66, 61, NaN, 15, 39, 43];
sumData = 0;
k = 0;
while (k < length(data))
    k = k + 1;
    if (~isfinite(data(k)))
        continue;
    end

    sumData = sumData + data(k);
end
```

---

Figure 3. Program to Sum all Valid Data

---

```

while (k < length(data))
    k = k + 1;
    if (isfinite(data(k)))
        sumData = sumData + data(k);
    end
end
end

```

---

Figure 4. While Loop Segment to Sum all Valid Data

### Radioactive Decay

The program of Figure 5 uses a flag controlled while loop to determine when a radioactive material has decayed to a safe level. The initial level is  $N_0$  and the safe level is  $N_{\text{safe}}$ . Radioactive decay can be described by the recursive equation  $N_k = N_{k-1}(1-\lambda)$  where  $\lambda$  is the decay rate,  $N_k$  is the concentration at step  $k$ , and  $N_{k-1}$  is the concentration at the previous step  $k-1$ . The decay rate is a value between 0 and 1, the closer to 1 the faster the decay rate.

---

```

% initial and safe levels
N0 = 100;
Nsafe = 5;
% decay constant
lambda = 0.4;

k = 0;
N = N0;
safe = false;
while (~safe)
    % new level at decay step k
    k = k + 1;
    N = (1-lambda)*N;
    if (N < Nsafe)
        safe = true;
    end
end
end

fprintf('It took %d steps to reach safe concentration.\n', k);

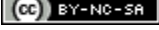
```

---

Figure 5. Program to Determine when Radioactive Material Decays to Safe Level

The flag `safe` is initially set to `false`. The loop control expression is true as long as the flag is `false`. Each iteration of the loop corresponds to a decay step. In the loop body, the level for the next decay step is determined and the new level is compared to the safe level. If the new level is safe, the flag is set to `true` and the loop will be exited when the loop control expressions is evaluated next.

Last modified Wednesday, March 30, 2022

 [MATLAB Marina](#) is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.