ENGR 105: Introduction to Scientific Computing

Variable and Function Naming Conventions, Accessing Vector Elements, Precedence, Iteration, Relational Operators

Dr. Graham. E. Wabiszewski

Function and array naming conventions

Functions and arrays (variables) have two naming requirements:

1) Consist of only the letters a-z, the digits 0-9, and underscores (_)
2) Must start with a letter

The filename of functions should adopt the function name

```
function output = myFunc(input1, input2)
```

would be saved as myFunc.m

Function and array naming conventions

Variable and function naming best practices (you don't have to follow these):

1) Low level functions start with lowercase letters (myFunc1, doThisThing)
2) High level functions start with uppercase letters (HW1prob1, SemesterProject)
3) Camel caps (capitalizing each instance of a word in a multi-word title / function name) can help distinguish words

   `takeinmatrixandtranspose` is much harder to distinguish than `takeInMatrixAndTraspose`

4) You can also use subscripts between names (ex. `take_in_matrix_and_transpose`)

Initializing vectors in MATLAB

You now know of several ways to input a vector into MATLAB:

- Discrete method

```
x = [0.5, 1.5, 2, 3.2, 1.5, 0.4];
```

- Colon operator - establishes vectors with user defined ranges and regularly-spaced intervals

```
x = 1:1:5; or x = 1:5;
x = -5:2:5;
```

- Linspace and logspace

```
linspace(1,5,5)
logspace(1,5,5)
```

Accessing vector elements

What about accessing vector elements?

For the following assume we have defined the following vector `x = [5, 9, 4, 1, 7, 3, 4, 8];`

- Access the first vector element

```
x(1) = 5
```

- Access a range of vector elements

```
x(1:3) = [5, 9, 4];
x(6:9) = [3, 4, 8];
```

- Access the last vector element

```
x(end) = 8;
```

Accessing vector elements

- Access the penultimate vector element

```
x(end-1) = 4;
```

- Access discrete elements

```
x([1 3 6]) = [5, 4, 7];
```

Removing vector elements

Vector elements can be removed by specifying an empty set "[]"

Say we establish the following vector

```
x = [8, 4, 5, 9, 3, 2, 5, 6];
```

- Removing the first element of the vector

```
x(1) = [];
x = [4, 5, 9, 3, 2, 5, 6];
```

- Removing several elements from the vector

```
x(end-3:end) = [];
x = [8, 4, 5, 9];
```

Mathematical / Operator Precedence in MATLAB

Precedence	Operator	MATLAB Representation
1	Transpose	A’
2	Parentheses	(A)
3	Power, left to right	A^2
4	Multiplication and division, left to right	A*2; A/2
5	Addition and subtraction, left to right	A+2; A-2
6	Colon operator	1:5

Precedence example: colon operator

The colon operator and addition

- The book shows the following example

```
x = 1 + 1:5;
  = [2, 3, 4, 5]
```

The colon operator takes the lowest precedence and the above call is equivalent to

```
x = 2:5; -> (1+1):5
```

- What would be the result of the following expression?

```
x = 1 + 2 + 1:5 + 6;
```

Overriding precedence

Operation precedence may be overwritten using parentheses

Consider the following examples - assume a, b, and c are variables

1) `a/b*c` -> statements are evaluated from left to right and the division is performed first

2) `(a/b)*c` -> the user has specified (through parentheses) that division should be performed first and this statement is equivalent to that above

3) `a/(b*c)` -> the user has specified (through parentheses) that multiplication should be performed before division

(1) and (2) are equivalent, (3) is not

“Over specifying” precedence for clarity

Parentheses can be included to increase the clarity of an expression even when not modifying precedence

(1)
```
z = 2*pi*x^2*y+1;
```

(2)
```
z = 2*pi*(x^2*y)+1;
```

(1) and (2) give the same result, but it is easier to see the relationships between pre-factors and independent variables in (2)

Iteration / for loops

Formula vectorization is preferred for speed, but sometimes iteration / loops are necessary

- A basic for loop that displays 1 through 20, by 1, at the command window

```
for jj = 1:20
    disp(jj)
end
```

- For loops are preferred over while loops (shown later) when the number of iterations are known / fixed

Iteration / for loops

- For loops can take in colon operators of any viable form

```
for jj = 1:3:19
    disp(jj)
end
```

- For loops can also take discrete vectors

```
for jj = [1 9 5]
    disp(jj)
end
```

Note: the for loop needs a closing "end" statement to execute properly

Iteration / for loops

- For loops can be nested

```
for jj = 1:10
    for kk = 1:10
        disp([jj kk])
    end
end
```

Note: each instance of the for loop requires an “end” statement

Formula vectorization is preferred for speed, but sometimes iteration / loops are necessary

- A basic for loop that displays 1 through 20, by 1, at the command window

```
for jj = 1:20
    disp(jj)
end
```

- For loops are preferred over while loops (shown later) when the number of iterations are known / fixed

An example of a for loop

Section 2.7.1 shows the use of a for loop for determining the square root of a number using Newton's method. Why is a for loop necessary here?

```
% Newton's method - as shown in section 2.7.1 of Essential Matlab, ed. 5,
% evaluated using for loops

clear; clc; close all

% We want to know the square root of scalar a
a = 2;

% Initial guess
x = a/2;

% For loop to converge to a solution
for jj = 1:10
    x = (x+a/x)/2;
end

disp(['The approximate square root is ',num2str(x,10),...
    ' and the actual solution is ',num2str(sqrt(a),10)])
```

Relational operators

Loops can also be invoked using the `while` command - however, we first need to understand relational operators since while loops depend heavily on them

Relational operators compare values (or vectors, matrices) and yield a true or false result

Relational Operator Syntax in Matlab

Relational Operator	Meaning
<	Less than
<=	Less than or equal to
==	Equal to
~=	Not equal
>	Larger than
>=	Larger than or equal to

Note: this is in error on pg. 66 of Essential Matlab, ed. 5

Iteration / while loops

The `while` command executes a piece of code until a desired condition is satisfied

```
while <relationalExpresion>
    <expressionToEvaluate>
end
```

A simple example

```
a = 1;
while a < 10
    a = a+1;
end
```

Iteration / while loops

While loops are most appropriate when the extent (number of) iterations is unknown a priori

(DEMO) Newton's method using a while loop to estimate the square root of a to a finite precision