Lecture 6

Lecture 6: Looping

- **Looping Statements**
  - Allow other statements to be repeated.
- **Two Basic Kinds of Loops**
  - Counted loops: repeat statements a specified number of times
  - Conditional loops: repeat statements, but ahead of time it is not known how many times the statements will be repeated
  - The statements that are repeated in the loop are called the action of the loop.
  - In practice, the **for** statement usually is used as the counted loop, and the **while** statement is used as the conditional loop

Lecture 6: Looping

- **The For Loop**
  - The for statement, or the for loop, is used when it is necessary to repeat statements in a script or function, and when it is know ahead of time how many times the statements will be repeated.
  - The variable that is used to iterate through values is called a loop variable, or an iterator variable
  - in MATLAB both i and j are built-in values for (-1)^0.5, so using either as a loop variable will override that value. If that is not an issue, then it is acceptable to use them as the loop variables.

Lecture 6: Looping

The For Loop (Cont)

- The general form of the for loop is

```
for loopvar = range
    action
end
```

Actually 'range' is a vector or a matrix. The loop variable iterates through the elements in this vector or matrix.

where loopvar is the loop variable, range is the range of values through which the loop variable is to iterate, and the action of the loop consists of all statements up the end.

Example:

```
>> for i = 1:5
   fprintf('%d\n', i)
   end
```

Please do the following experiments:

1) for i=1 : 2 : 5
2) for i=0.1 : 0.2 : 0.5
3) for i=[0.1, 0.2, 0.5]
4) for i=[1 2 3; 4 5 6]

See what will happen

Lecture 6: Looping

Finding Sums and Products

- One common application of for loop is to calculate sums and products
- Example: Calculate the sum of the integers 1 through n: 1 + 2 + 3 + … + n

sum_1_to_n.m

```
function runsum = sum_1_to_n(n)
% This function returns the sum of integers from 1 to n
runsum = 0;
for i = 1:n
    runsum = runsum + i;
end
```

First, the sum has to be initialized to zero.

Add every value to the sum

```
>> sum_1_to_n(5)
ans =
    15
```

Lecture 6: Looping

- **Finding Sums and Products**
  - Example:

    Calculate the product of the integers 1 through n: 1 * 2 * 3 * 4 * … * n

    myfact.m

```
function runprod = myfact(n)
% This function returns the product of integers from 1 to n
runprod = 1;
for i = 1:n
    runprod = runprod * i;
end
```

First, the product has to be initialized to one.

Multiply the product with every value

```
>> myfact(5)
ans =
     120
```

The efficient method: MATLAB has a built-in function, **factorial**, that will find the factorial of an integer n.

```
>> factorial(5)
ans =
   120
```

Lecture 6: Looping

- **Sums and Products with Vectors**
  - Example: Calculate the sum of elements in a vector.

    myvecsum.m

```
function outarg = myvecsum(vec)
% This function sums the elements in a vector
outarg = 0;
for i = 1:length(vec)
    outarg =  outarg + vec(i);
end
```

```
>> myvecsum([5 9 4])
ans =
    18
```

The programming concept: The vector is passed as an argument to the function. The function loops through all the elements of the vector, from 1 to the length of the vector, to add them all to the running sum.

  - The efficient method: sum(vec)

    Example:

```
>> sum([5  9  4])
>> ans =
        18
```

Questions: How to calculate the sum of some particular elements in a vector?

Answer: Put the element indices in a vector and use this vector as the range for loop variable

Lecture 6: Looping

- **Sums and Products with Vectors**
  - Example: Calculate the product of elements in a vector.

    myvecprod.m

```
function outarg = myvecprod(vec)
% This function finds the product of the elements in a vector
outarg = 1;
for i = 1:length(vec)
    outarg = outarg * vec(i);
end
```

```
>> myvecprod([5  9  4])
ans =
      180
```

  - The efficient method: prod(vec)

    Example:

```
>> prod([5  9  4])
>> ans =
          180
```

Lecture 6: Looping

- **Combining for Loops with if Statements (if, if-else, nested if, elseif)**
  - Example: to find the minimum value in a vector.
  - The programming concept
    - ✓ The minimum so far is the first element in the vector
    - ✓ Loop through the rest of the vector

myminvec.m

```
function outmin = myminvec(vec)
% Finds the minimum value in a vector
outmin = vec(1);
for i = 2:length(vec)
    if vec(i) < outmin
        outmin = vec(i);
    end
end
```

```
>> vec  =  [3   8   99   11];
>> myminvec(vec)
ans =
      3
```

If statement is used to determine whether another statement is executed or not.

- The Efficient Method

  MATLAB has built-in functions min and max, which find the minimum and maximum values in a vector

```
>> vec = [5  9  4]
>> min(vec)
ans =
     4
```

Lecture 6: Looping

➢ Input in a for Loop

- Example: repeat the process of prompting the user for a number, store them into a vector, and print the number.

‘n’ is the number of times to repeat the process.

forinputvec.m

```
function numvec = forinputvec(n)
% Prompts the user and puts the number into a vector
numvec = zeros(1,n);
for iv = 1:n
    inputnum = input(‘Enter a number: ’);
    numvec(iv) = inputnum;
    fprintf(‘You entered %.1f\n’, inputnum)
end
```

Create a 1xn vector with zeros.

The statement ‘numvec=zeros(1,n)’ is not necessary, because the vector can be extended when assigning values to it with ‘numvec(iv)=inputnum’

```
>> myvec = forinputvec(3)
```

Lecture 6: Looping

- **Nested For Loops:**
- The action of a loop can be any valid statements. When the action of a loop is another loop, this is called a nested loop.

  Example: write a function multtable calculates and returns a matrix that is a multiplication table. Two arguments are passed to the function, which are the number of rows and columns for the matrix. (multiplication table: the element at i-th row and j-th column is i*j)

multtable.m

```
function outmat = multtable(rows, columns)
% Creates a matrix which is a multiplication table

% Preallocate the matrix
outmat = zeros(rows, columns) ;
for i = 1:rows
    for j=1:columns
            outmat(i,j)=i*j ;
    end
end
```

'rows' and 'columns' are the numbers of rows and columns, respectively.

Create a 'row x column' matrix with zeros.

```
>> multtable(3,5)
```

Lecture 6: Looping

- **Nested Loops and Matrices:**
- Nested loops often are used when it is necessary to loop through all the elements of a matrix.

Example: calculate the overall sum of the elements in a matrix.

mymatsum.m

```
function outsum = mymatsum(mat)
% Calculates the overall sum of the elements

% in a matrix
[row  col] = size(mat);
outsum = 0;
for i = 1:row
      for j=1:col
               outsum = outsum + mat(i,j);
      end
end
```

size(mat) returns the numbers of rows and columns in a vector.

```
>> mat = [3 : 5;  2  5  7]
mat =
      3   4   5
      2   5   7
>> mymatsum(mat)
ans =
      26
```

- The Efficient Method

  MATLAB has a built-in function sum, which will return the sum of each column

```
>> sum(mat)
ans =
      5   9   12
>> sum(sum(mat))
ans =
      26
```

Lecture 6: Looping

- **Summary**
  - Use for loop to find sums and products
  - Combine for loops with if statements
  - Call input function in a for loop
  - Nested for loops

- **Homework on Canvas (Due next Wednesday)**