

MATLAB II

Computational Physics

MATLAB (you need to know) Lecture 2

Outline

- MATLAB Scripts (or Programs)
 - Structure
 - Comments
 - Initialization
 - Calculation

Program Structure

- Comments
- Initialization
 - Define and Initialize Variables
- Calculation
 - Carry out calculation
- Display Results
 - Plot Results
 - Write results to file

fall.m

```
% falling ball demo script -- fall.m
% Author: F. P. Schloerb
% computes solution of falling object under force of gravity
%  $x(t) = x(0) - 1/2 g t^2$ 
%
```

Comments

```
% define an array of times for calculation
% array t spans times from 0 to 10 seconds in steps of 1s
t = 0:1:10;
```

Initialization

```
% get the initial height from the user:
h = input('Enter initial height (m): ');
```

```
% compute the array of positions corresponding to array of times,
t
% note use of .* operator for element-by-element multiplication of
t arrays
x = h - 0.5 * 9.8 * t.*t;
```

Calculation

```
% plot the result - with labels for axes
plot(t,x);
xlabel('time (s)');
ylabel('position (m)');
title('Falling Ball');
```

Display Results

```
% display results in a table that looks nice
fprintf('Here are results:\n')
fprintf(' t      x\n')
fprintf(' ---  -----\n')
for i=1:11
    fprintf('%5.1f %10.2f\n',t(i),x(i))
end
```


Comments

- Comments are useful for explaining what program is doing, both to yourself and others.
- In MATLAB, comments follow the % character
- Examples:

```
% this is a comment line
```

```
x = y * 2; % this comment follows a statement
```

Comments on Comments

- Use comments liberally
 - Others (e.g. graders) won't know what you are doing without comments.
 - You won't remember what you were doing when you look at the program in the future.
- Comments must be useful. Consider....

```
% initialize t  
t = 0:1:10;
```

vrs.

```
% initialize array of times for calculation; time in s.  
t = 0:1:10;
```

Initialization Methods

- Arrays
 - colon operator
 - special functions fill arrays
 - zeros - initialize w/0
 - ones - initialize w/ 1
 - load from data file containing numbers in rows and columns

```
T = 0:1:10;
```

```
x = zeros(2,4);  
y = ones(4,4);
```

```
load my_file.dat;
```

Initialization Methods

- Assignment
 - direct assignment to variable
 - element by element assignment
- User Input

```
z = 5;
```

```
t = zeros(1,10);  
for i=1:10  
    t(i) = 3.0;  
end
```

```
% ask the user  
h = input('enter h');
```

Logical Variables and Statements

- Logical variables evaluate to 1 (true) or 0 (false)
- Logical Operators:
 - comparison:
< <= > >= == ~=
 - Logical 'and' (&)
 - Logical 'or' (|)
- Primary use in conditional execution

```
% define variables  
a = 1;  
b = 0;  
  
% sample operators:  
% greater than  
c = a > b;           % c = 1  
% is equal to  
d = a == b;         % d = 0  
% logical or  
e = a | b;          % e = 1  
% not equal to  
f = a ~= b;         % f = 1
```

Calculation

Conditional Execution

- if statement allows for conditional execution of another statement.

- Syntax:

```
if (logical-statement-1)  
    execute-statement-1;  
elseif (logical-statement-2)  
    execute-statement-2;  
else  
    execute-statement-3;  
end
```

```
% a simple example
```

```
if(x>5)
```

```
    y = 1;
```

```
end
```

```
% more complicated
```

```
if( x == 0 | x == 1)
```

```
    flag = -1;
```

```
elseif(x < 0 & x ~= -1)
```

```
    flag = -1;
```

```
else
```

```
    flag = 0;
```

```
end
```

Calculation

Iteration of Arrays

- Most of our problems involve *iterating* through MATLAB arrays and working with the elements.
 - for loop iteration
 - while loop iteration

```
% for loop example  
for i=1:10  
    x(i) % just prints  
end
```

```
% while loop example  
% initialize i to start  
i = 1;  
while (i<10)  
    % increment i  
    i = i+1;  
    x(i) % just prints  
end
```

A Useful Function

length

- MATLAB provides a useful function:
length
- For a 1-D array t the length function returns the number of elements in the array: e.g. if $t = [1\ 2\ 3\ 4\ 5]$
then $\text{length}(t) = 5$
- This is extremely useful for writing scripts where the number of values of the independent variable may change.

length

Example from Command Line

```
>> c = [0 1 2 3 4]
```

Define vector c

```
c =  
  0  1  2  3  4
```

```
>> length(c)
```

Use length function to find number of elements in c

```
ans =  
  5
```

```
>>
```

Iterating Arrays

Example

t is an array going from 0 to 1 spaced by 0.1

t = [0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0]


this is t(1)


this is t(5)


this is t(11)

Operation creates a new array: x

x = [0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0]

i is scalar used to count through array elements

*first time in loop i=1 so: y(1) = 10.0*t(1)*

*next pass has i=2 so: y(2) = 10.0*t(2)*

etc. until i = last element in t array

final result is the new array: y

y = [0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0]

```
% create array t  
t = 0.0:0.1:1.0;
```

```
% use t to create x  
x = 10.0*t ;
```

```
% element by element  
% operation in for loop
```

```
for i=1:length(t)  
    y(i) = 10.0*t(i);  
end
```

```
/home/schloerb/fall.m*
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
x =
0 1 4 9 16 25 36 49 64 81 100
x =
0 1 4 9 16 25 36 49 64 81 100
% 0 1 4 9 16 25 36 49 64 81 100
plot(t,x);
xlabel('time (s)');
ylabel('position (m)');
title('Falling Ball');
% display results in a table that looks nice
fprintf('Here are results:\n')
fprintf(' t x\n')
fprintf(' --- ----\n')
```

Place cursor on variable name to view array contents

x =	0	1	4	9	16	25	36	49	64	81	100
-----	---	---	---	---	----	----	----	----	----	----	-----