MATLAB I

Computational Physics

MATLAB
(you need to know)
Lecture 1
Outline

• MATLAB Mathematics
  • Variables
  • Mathematical operations
  • Colon Operator

• About Programs
  • Comments
  • Initialization
  • Calculation

• The MATLAB “Environment”
Command Line Input

*Help*

- help followed by command or function name will give information about that command.

- Examples:

  help fprintf - produces help information for fprintf, which is used later in this lecture.

  help plot - produces help information on the plot function
MATLAB Variables

- Fundamental Data Type: Matrix

\[
C = \begin{bmatrix}
0 & 1 & 1 \\
2 & 3 & -1 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

- Special Cases
  - 1x1 Matrix = scalar
  - 1xN Matrix = row vector
  - Nx1 Matrix = column vector

\[
Z = 1;
\]
\[
a = [0 \ 3 \ 4];
\]
\[
b = [1; \ 2; \ 3];
\]
The *length* Function

- 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 `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]
c =
0 1 2 3 4
>> length(c)
an =
5
>>
```
MATLAB

Mathematics Operations

- All matrices defined as they are used; no strict typing as in C.

- Operations
  - +  -  *  / are matrix operations.
  - Special "." operation works element-by-element: ( .+  .-  .*  ./ )

- ' (apostrophe) operator: Hermitian Conjugate (transpose of complex conjugate)

- .' operator: Transpose of Matrix
Matrix Operations

Examples

```matlab
>> t = [1 2 3 4 5];
>> z = t*t;
??? Error using ==> *
Inner matrix dimensions must agree.
```

- Define Row Vector
- Try to multiply row x row ... doesn't work!!

```matlab
>> w = [1; 2; 3; 4; 5];
>> z = t*w
```

- Make Column Vector
- row x column works! (NOTE: no ; at end of line causes result to print)

```
z =
   55
```

```matlab
>> x = t.*t
```

- example of .* operation

```
x =
   1  4  9 16 25
```
The Colon Operator

- Define a vector:
  \[ t = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10] \]

- Define
  \[ x = [0\ 0.2\ 0.4\ 0.6\ 0.8] \]

- Access a row or column in a matrix:
  \[ A = [1\ 2\ 3; 4\ 5\ 6; 7\ 8\ 9] \]
  \[ \text{first\_row} = [1\ 2\ 3] \]
  \[ \text{first\_col} = [1; 4; 7;] \]
MATLAB clear Command

- MATLAB creates new vectors and matrices as you "request" them.
- These remain in the environment until you quit MATLAB.
- Sometimes it is necessary to enter clear command to reinitialize.

```plaintext
>> a = 1:2
a =
   1  2
>> a(5) = 12
a =
   1  2  0  0  12
>> clear
>> a
??? Undefined function
>> a = 1:2
a =
   1  2
```
MATLAB M-Files

- M-Files allow us to carry out tasks requiring many steps using MATLAB.
- Two types of M-Files in MATLAB
  - **Script**: a file containing a sequence of MATLAB statements executed one at a time as if typed at the keyboard.
  - **Function**: a file containing the definition of a new MATLAB function with specific inputs and outputs.
- `fall.m` is an example of a script.
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
% \[ x(t) = x(0) - \frac{1}{2} g t^2 \]
%
% define an array of times for calculation
% array t spans times from 0 to 10 seconds in steps of 1s
% \[ t = 0:1:10; \]
% get the initial height from the user:
% \[ h = \text{input}('\text{Enter initial height (m): }'); \]
% compute the array of positions corresponding to array of times, t
% note use of .* operator for element-by-element multiplication of t arrays
% \[ x = h - 0.5 * 9.8 * t.*t; \]
% plot the result - with labels for axes
% \[ \text{plot}(t,x); \]
% \[ \text{xlabel}('\text{time (s)}'); \]
% \[ \text{ylabel}('\text{position (m)}'); \]
% \[ \text{title}('\text{Falling Ball}'); \]
% display results in a table that looks nice
% \[ \text{fprintf}('\text{Here are results:}\n'); \]
% \[ \text{fprintf}('\text{' t x}\n'); \]
% \[ \text{fprintf}('\text{--- ------}\n'); \]
% for \[ i=1:11 \]
% \[ \text{fprintf}('\%5.1f %10.2f}\n',t(i),x(i)) \]
% end
Comments

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

```matlab
% 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....

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

*vrs.*

```matlab
% 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:

```plaintext
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
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: \( y \)

\[ y = [ 0.0 \ 1.0 \ 2.0 \ 3.0 \ 4.0 \ 5.0 \ 6.0 \ 7.0 \ 8.0 \ 9.0 \ 10.0 ] \]
The MATLAB Environment

• Our version of MATLAB has extensive Graphical User Interface

• Features:
  • Command Entry
  • View of all Variables
  • On line Help
  • Editing M-Files
  • Running and Debugging
  • Profiling
Command Window: command entry and results

Workspace Window: view/edit variable/arrays
Useful Windows

- **Command**: Issue MATLAB commands and view printed results
- **Workspace**: View arrays that are present in memory
- **Help**: help with MATLAB commands
- **Editor/Debugger**: Write, Edit, Save, Run, Debug MATLAB scripts.
- **Profiler**: Measure time consumed in program steps.
Select M-File to Edit Scripts

Other Useful Features:
- Import Data (works like load command)
- Save Workspace
% falling ball demo script -- fall.m

% computes solution of falling object under force of gravity
% \( x(t) = x(0) - \frac{1}{2} g t^2 \)
%

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

% get the initial height from the user:
% \( h = \text{input('Enter initial height (m): ')}; \)

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

% plot the result - with labels for axes
% \( \text{plot}(t,x); \)
% \( \text{xlabel('time (s)');} \)
% \( \text{ylabel('position (m)');} \)
% \( \text{title('Falling Ball');} \)
The script calculates the position of a falling object under the force of gravity. It prompts the user for the initial position, initial velocity, and time step, and calculates the position at each time step, storing the results in arrays. The final position is plotted with respect to time.
% get the
h = input;

% compute
x = h - 0

% note use

t = array of times, t

% multiplication of t arrays

% display results in a table that looks nice
fprintf('Here are results:
')
fprintf('
')
fprintf('
')
fprintf('
')
fprintf(' t x
')
fprintf('--- ------
')
for i=1:11
 fprintf('
')
 fprintf('
')
 fprintf('
')
 fprintf('
')
 fprintf('
')
 fprintf('%5.1f %10.2f', t(i), x(i))
end
Select Run or Save and Run to run Script from Editor Window.
% get the initial height
h = input('Enter initial height, h: ');

% compute the array of positions
x = h - 0.5 * 9.8 * t.*t;

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

% display results in a table that looks nice
fprintf('Here are results:
');
fprintf(' t   x
---   ---
');
for i=1:11
    fprintf('%5.1f %10.2f
',t(i),x(i))
end
% array t spans times from 0 to 10 seconds in steps of 1s
\[ t = 0:1:10; \]

% get the initial height from the user:
\[ h = \text{input('Enter initial height (m): ')}; \]

% compute the array of positions corresponding to each
% note use of .* operator for element-by-element multiplication of \( t \) arrays
\[ x = h - 0.5 \times 9.8 \times t \times t; \]

\[
\begin{array}{cccccccccccc}
0 & 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 & 100 \\
\end{array}
\]

plot(t,x);
xlabel('time (s)');
ylabel('position (m)');
title('Falling Ball');

% display results in a table that looks nice
fprintf('Here are results:
');
fprintf(' t \times x
');
fprintf(' --- --------
');

UNDOCKED

DOCKED