MATLAB Functions and I/O

ENGR 151, Lecture 19: 17 Nov 14

Announcements

- Project 6 graded
- Project 7 out (mostly), due Wed 26 Nov
  - See spec and visit Cody site

Input/Output Question

Suppose we execute the script at right, and enter

```
1 + x/100
```

at the prompt.

What is output?

```
x = 38;
y = input('Enter a number: ');
disp(y);
```

A 1.3800

B 1

C 1 + x/100

D nothing

E Error: number expected

Input/Output Question ('s' option)

Suppose we execute the script at right, and enter

```
1 + x/100
```

at the prompt.

What is output?

```
x = 38;
y = input('Enter a number: ','s');
disp(y);
```

A 1.3800

B 1

C 1 + x/100

D nothing

E Error: number expected

Input/Output Question (fprintf)

Suppose we execute the script at right, and enter

```
1 + x/100
```

at the prompt.

What is output?

```
x = 38;
y = input('Enter a number: ');
fprintf('%.0f', y);
```

A `1.3800`

B `1`

C `1 + x/100`

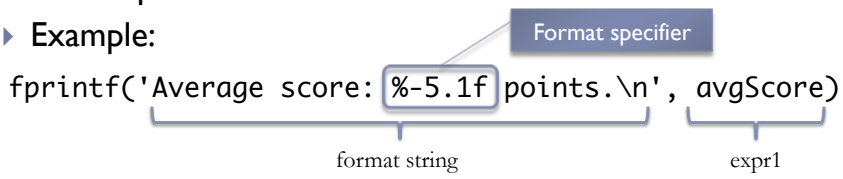
D nothing

E Error: number expected

Formatted Output

- The procedure `fprintf` provides fine control of how data is formatted for output to display or file
- General form for display output:

  `fprintf(`*format_string*`,` *expr1* `,` *expr2*`, ...)`
  - displays expression values as specified in the format string.
- Format string: literal string, with specifications to format each expression
- Example:



Format Specifiers

- General form for format specifier:

  %{*flag*}{*width*}{ . {*precision*}}*conversion*

- flag
  - +: include sign (+/–)
  - −: left justify
  - 0: pad with zeros
- width: field width (number of characters)
- precision: places after decimal point
- conversion: code for format option (*next…*)

Some Conversion Codes

- Numeric
  - %d integer number
  - %e exponential notation (lowercase e)
  - %E exponential notation (uppercase E)
  - %f fixed point notation
  - %u integer (unsigned)
- Alphabetic
  - %c single character
  - %s string

Some Special Characters

- Use backslash (\) for escape sequence
  - \t tab
  - \n new line
  - \\ backslash
  - \% percent

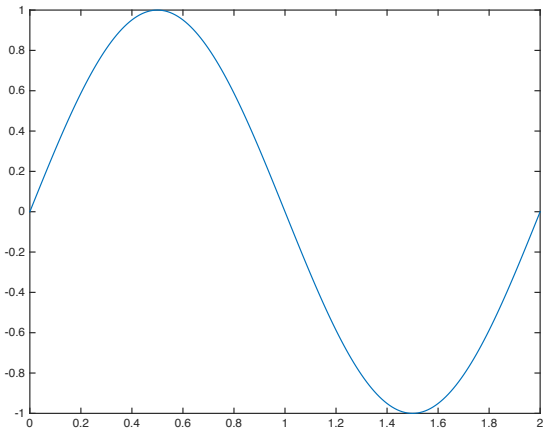
fprintf Examples

```
num = 1234.56789;

fprintf('The number is %f !\n',num);
 The number is 1234.567890 !

fprintf('The number is %e !\n',num);
 The number is 1.234568e+03 !

fprintf('The number is %.1f !\n',num);
 The number is 1234.6 !

fprintf('The number is %08.1f !\n',num);
 The number is 001234.6 !
```

Applying a Scalar Function to an Array

- What is the value of `sin(0:pi/2:2*pi)`?

A. Error
B. `0`
C. `[0 1 0 -1 0]`
D. `[-1 1]`
E. None of the above



Scalar Functions Apply Elt-by-Elt to Arrays

- `sin(0:pi/2:2*pi)` → `[0 1 0 -1 0]`
- `char(65:70)` → `ABCDE`
- `mod(1:10, 3)` → `[1 2 0 1 2 0 1 2 0 1]`
- `mod(10, 1:3)` → `[0 0 1]`
- `sin(eye(3)*pi/2)` → `[1 0 0; 0 1 0; 0 0 1]`

Applying a Scalar Function to Two Arrays

- What is the value of `mod(10:13, 1:4)`?

A. Error
B. `[0 1 0 1]`
C. `[0 0 1 2; 0 1 2 3; 0 0 0 0; 0 1 1 1]`
D. `[0 0 0 0; 0 1 0 1; 1 2 0 1; 2 3 0 1]`
E. None of the above

Trigonometric Functions

- `sin(x)` sine, x in radians
- `cos(x)` cosine, x in radians
- `tan(x)` tangent, x in radians
- `asin(x)` arcsine in radians
- `acos(x)` arccos in radians
- `atan(x)` arctan in radians
- `atan2(y,x)` arctan(y/x) over all quadrants

Other Mathematical Functions

- `abs(x)` absolute value
- `angle(x)` phase angle of a complex number
- `exp(x)` e^x
- `log(x)` natural logarithm
- `mod(x,y)` modulo function (remainder)
- `sqrt(x)` square root

Rounding Functions

- `ceil(x)` Ceiling (rounds up)
- `fix(x)` Rounds toward zero
- `floor(x)` Rounds down
- `round(x)` Rounds to nearest integer

String Conversions

- `char(x)` Converts numbers to characters using ASCII
- `double(x)` Converts character to number
- `int2str(x)` Converts `round(x)` to a string
- `num2str(x)` Converts a real number into a character string with a decimal
- `str2num(s)` Converts a string into a number

Vector Functions

- `length(v)` number of elts
- `min(v)` minimum elt
- `max(v)` maximum elt
- `sum(v)` sum over elts
- `prod(v)` product over elts
- `mean(v)` arithmetic average
- `median(v)` median
- `norm(v,p)` p-norm of vector
  `sum(abs(v).^p)^(1/p)`

Defining MATLAB Functions

- MATLAB supports *procedural abstraction*
- Define functions within .m files
- Function definition format:

```
function [outputs] = function_name (inputs)
% H1 line: function name and short description
% comments documenting function behavior
%

Body
```

Function Parameters and Return Values

```
function [outputs] = function_name (inputs)
% H1 line: function name and short description
% comments documenting function behavior
%

Body
```

- inputs: comma-separated list of argument variable names
  - variables passed by value
- outputs: list (comma- or space-separated) of output variable names
  - variables must be assigned in function body
  - on termination these values returned
- All names in function have local scope
  - Function cannot reference or modify variables in workspace scope

C++ Function and Corresponding MATLAB Function

```
double MyFun( double a, double b ){
     return a*b*b;
}
```

```
function c = MyFun( a, b )
     c = a*b*b;
end
```

Handling Vector Inputs

```
double MyFun( double a, double b ){
     return a*b*b;
}
```

Out of luck

```
function c = MyFun( a, b )
     c = a.*b.*b;
end
```

Now works

C++ Procedure and Corresponding MATLAB Function

```
void PrintThem( double a, double b ) {
     cout << a << endl;
     cout << b << endl;
     return;
}
```

```
function PrintThem( a, b )
     disp(a);
     disp(b);
end
```

C++ Procedure and Corresponding MATLAB Function

```
void means(double a, double b,
              double & c, double & d) {
     c = sqrt(a*b);
     d = 0.5*(a+b);
     return;
}
```

```
function [c, d] = means(a, b)
     c = sqrt(a.*b);
     d = 0.5*(a+b);
end
```

Exercise

What is the MATLAB equivalent of this C++ procedure?

```
void process (vector <double> list, double & g,
              vector <double> & h) {
  for (int x=0; x<list.size(); x++) {
     h[x] = list[x]*3.0;
     if (x==0 or g < h[x]) g = h[x];
  }
}
```

```
function [g, h] = process( list )
% sorry I cannot help you

   h = 3 * list;
   g = max(h);
```

Exercise: Convert from MATLAB to C++

```
function [x, y] = SplitString(a)
  s = floor(length(a)/2) ;
  x = a(1 :s) ;
  y = a(s+1:length(a));
```

```
void SplitString(const string & a,
                        string & x, string & y)
{
  int s = a.size()/2;
  x = a.substr(0,s);
  y = a.substr(s,a.size()-s);
  return;
}
```

Subfunctions

- A function (.m) file may contain multiple function definitions
  - The first is the primary function and should correspond to the name of the file
  - Subsequent functions are called subfunctions
- Primary and subfunctions may call each other
  - regardless of order of appearance in file
  - only primary function may be called from main scope
- Each (sub)function has its own scope for local variables

Example: What is Output?

```
function [a,b] = FindStuff(A)
a = CountThis(A);
b = CountThat(A);

function count = CountThis(A)
count = sum(mod(A,2))

function count = CountThat(A)
a = 4
count = sum(~mod(A,3))
```

```
>> a=[1 2 3 4 5 6 9];
>> [this,that] = FindStuff(a)
```

this ?
that ?

A	B	C	D	E
1	2	3	4	5

Example MATLAB Program

- Write a program that:
  - prompts the user for the initial height and velocity of a ball at time zero and a duration.
  - plots the height and velocity of the ball over the duration.

$v = g\,t + v_0$

$h = g\,t^2/2 + v_0\,t + h_0$

Gravity Program (comments)

- Start by commenting your program
  - comments should include a glossary of identifiers
- Comment individual segments of code as well

```
function gravity()
% gravity computes, plots
the trajectory of a ball
%
% g = gravitational accel
% x = positions of ball
% x0 = initial position
% v = velocities of ball
% v0 = initial velocity
% t = time
% tmax = duration

% set gravitational accel
g = -9.8;
```

Gravity Program (input, calcs)

- Get input from user
- Make time vector
- Calculate velocity and position

```
% get input from user
x0 = input('Height: ');
v0 = input('Velocity: ');
tmax = input('Duration: ');

% set up time
t = 0:tmax/100:tmax;

% calculate velocity
v = v0 + g * t;

% calculate position
x = x0 + v0 * t ...
    + 0.5 * g * t .^ 2;
```

Gravity Program (plot)

- Get input from user
- Make time vector
- Calculate velocity and position
- Finally, plot results

```
% set up time
t = 0:tmax/100:tmax;

% calculate velocity
v = v0 + g * t;

% calculate position
x = x0 + v0 * t ...
    + 0.5 * g * t .^ 2;

% plot position, velocity
plot(t,x,'r-',t,v,'b:');
xlabel('time (s) ');
ylabel('h(m); v(m/s)');
legend('ht','velocity');
grid on;
```

Save/Load

- MATLAB provides facilities to save some or all of the current workspace state for later use.
- `save mydata`
  - Stores all data in current workspace variables to a file named `mydata.mat`
  - binary format readable only by MATLAB
- `load mydata`
  - Restores all data from `mydata.mat` to current workspace

Save/Load Selected Variables

- `save mydata x y {...}`
  - Stores data in specified variables to `mydata.mat`
- `load mydata x y {...}`
  - Restores data from specified variables from `mydata.mat` (if present) to current workspace

Save/Load Procedure Syntax

- `save(filename,var1,...)`
  - Example: `save('mydata','x','y')`
  - Stores data in specified variables to file with specified filename (using .mat as default extension)
- `load(filename,var1,...)`
  - Restores data from specified variables from specified file (if present) to current workspace

Save to ASCII file

- `save -ascii mydata.txt` {vars...}
- `load` assumes ASCII format unless extension is .mat
- Limitations
  - Data must be (0/1/2)-dimensional character or data array
  - All data is written as numbers
  - Imaginary part of complex numbers is lost
  - Variable names not saved
  - If using load to read data, all items must have the same number of columns

General File I/O

- MATLAB manages file input/output through file IDs
  - Numbers for tracking files used by a program
- To open a file:

```
file_id_var = fopen(filename);
file_id_var = fopen(filename,permission);
```

- Example:

```
fid = fopen('Myfile.txt', 'r');
```

returns -1 if file open fails

- To close this file:

```
close(fid)
```

Permissions (file open modes)

- `'r'` read only (default)
- `'w'` write to new or existing file
- `'a'` append to new or existing file

Type "help fopen" for full list of permissions

Formatted File I/O

- Use `fprintf` to output to files (optional first argument)

```
fid = fopen('Myfile.txt', 'w');
fprintf(fid, 'It is %.2f meters long', num);
```

- Analogous function called `fscanf` used to read formatted data

```
decnum = fscanf(fid, '%d');
```