Applied Engineering Problem Solving

Lesson #3: Programming in Matlab

Prof. John R. White
Chemical and Nuclear Engineering
UMass-Lowell, Lowell MA

Lesson #3 Goals

Additional Programming Features

The use of function subprograms for developing well-structured programs

Controlling the flow of a program via conditional tests and looping structures

Implementation of discrete formulas

Processing/documenting input and output data within Matlab (including proper internal documentation)

Developing proper programming logic and problem solving strategies (via example)…

Gilat:
Chapters 1 – 7 & 10

Chapra:
Chapters 1 – 3

Lesson # 3 Lecture Notes and Illustrative Examples

Functions…

Script file:

-- list of Matlab commands
-- no explicit input or output (in a manner of speaking)
-- all variables are stored in the Matlab workspace
-- data in workspace can be accessed within the command window or by another script file
-- can call another script file

Function file:

-- also contains list of Matlab commands
-- first executable line must start with the word "function"
-- all variables within the function are local to that function (not available from the command window)
-- communicates to main program via input and output argument list or via global variables
-- allows for development of modular well-structured programs

Both file types use the same *.m naming convention

Functions… (cont.)

Basic Syntax:

function [output arguments] = function_name(input arguments)

(do something useful with the input arguments, being sure to define the output arguments)

where the function… line must be the first executable line within the function file

The best way to illustrate the use of functions is via example:

In the formal notes, there is a series of examples using the functions fxy.m, fst.m, and fxyz.m to work with the simple math relationship:

$$f(x,y) = (x+2)y^2$$

You should spend a few minutes reading these notes!!!

Here, in class, we will do a similar set of manipulations using the mathematical relation:

$$f(t) = e^{at}\sin\omega t \qquad (a \text{ and } \omega \text{ are parameters})$$

Functions… (cont.)

$$f(t) = e^{at} \sin \omega t$$

where a = decay/growth factor and ω = angular frequency

and ω = 2πf with f = 1/T and T = time per cycle

The file name is fun1.m

```
function f = fun1(t,a,w)
f = exp(a*t).*sin(w*t);
```

a & w are scalars
t is a vector

This function is called by fun1_main.m to perform various tasks.

In addition, fun1_anon.m treats the same example with f(t) incorporated as an anonymous function instead of a separate file.

Let's do it in Matlab…

Note: This example is visited again after we discuss the use of conditional tests within Matlab…

Program Flow Control

Control of program flow is an essential ingredient of any full-featured programming language.

The ability to repeat mathematical evaluations or groups of commands many times is also a necessary feature.

Within Matlab,

flow control: *if … else … end* or *switch … end* structures

repeat operations: *for … end* or *while … end* constructs

In most cases, the *for … end* looping structure is used when some fixed number of loops is desired, whereas the *while … end* syntax is used when the loop is repeated until some test or condition is no longer valid.

Note also that the *switch … end* structure is sometimes useful as an alternative to a long sequence of *elseif* blocks within an *if … elseif … else … end* structure.

Conditional Tests

Most of these programming features -- the *if … else … end*, *switch … end*, and *while … end* structures, require some mechanism for determining whether a conditional test is true or false.

If the test it true, the loop is continued or the code segment within the *if … else … end* structure is executed.

If, however, the test is not satisfied, then program flow proceeds to the next *elseif* or *else* condition or to the *end* of the structure.

The tests are performed with a series of relational and logical operators, including an equality test, ==, a less than or equal to test, <=, the not equal to test, ~=, etc., etc...

see your texts or the Matlab *help* facility for a full list and for a discussion of operator precedence

Conditional Tests (cont.)

As a simple example, consider the following Matlab code:

```
>> format compact
>> x = [1 2 3 4 5];   y = [1 -2 3 -4 5];
>> z = x == y
z =
     1     0     1     0     1
>> z2 = x ~= y
z2 =
     0     1     0     1     0
>> z3 = x > y
z3 =
     0     1     0     1     0
>> z4 = x >= y
z4 =
     1     1     1     1     1
```

The result of the test is a logical array filled with 0 (false) or 1 (true) values

Conditional Tests (cont.)

It is important to note that a conditional test with a logical array returns a single logical variable that is true only if all the elements are true.

This is a biggie!!!

For example, the following code displays the second statement since the `if z == 1` statement fails (recall `z = [1 0 1 0 1]`),

```
>> if z == 1
     disp('x and y are identical')
   else
     disp('Not all corresponding values are equal')
   end
>> Not all corresponding values are equal
```

All the elements of the logical array must be true for this test to be true

This means that, in most cases, you will want to check on individual elements of a logical array within a looping structure!!!

Let's show an example in Matlab…

Conditional Tests (cont.)

To do this, let's revisit the example with the decaying sinusoid:

$$f(t) = e^{at} \sin \omega t$$

However, for this case, we will allow the decay factor, a, to change discontinuously at t = 5 sec, or

$$a = \begin{cases} -0.2\ s^{-1} & \text{for} \quad t \le 5\ s \\ -0.5\ s^{-1} & \text{for} \quad t > 5\ s \end{cases}$$

This capability is implemented and tested within the following files:

- fun2_main.m -- main program to call the function files and plot f(t)
- fun2a.m -- function file to evaluate f(t) within looping structure
- fun2b.m -- function file to evaluate f(t) using a vector approach

Let's show this in Matlab…

for … end vs. *while … end* loops

In most cases, the *for … end* looping structure is used when some fixed number of loops is desired, whereas the *while … end* syntax is used when the loop is repeated until some test or condition is no longer valid.

As an example, consider the evaluation of the following discrete expression:

$$f = \sum_{k=1}^{M} (k-2)^2$$

Case 1: What is the value of f for M = 30?

```
f = 0;  M = 30;
for k = 1:M
  f = f + (k-2)^2;
end
f
```

must initialize variables

add k^{th} term to running sum

for M known, a *for … end* loop that is indexed from 1 to M makes perfect sense…

for … end vs. *while … end* loops

$$f = \sum_{k=1}^{M} (k-2)^2$$

Case 2: What is the minimum value of M such that f > 5000?

```
ff = 5000;  f = 0;   k = 0;
while ff > f
  k = k+1;
  f = f + (k-2)^2;
end
M = k
```

must initialize variables

here we need to increment the index k

add k^{th} term to running sum

for unknown M, a *while … end* loop that is continued until the desired condition is met makes perfect sense…

Let's do these in Matlab (see series_1.m) …

Calculation of Error

Error based on a known value:

$$\varepsilon = \frac{\text{calculated - known}}{\text{known}}$$

if abs(ε) < tol the calculation is done

Error based on successive estimates:

Often we do not know the exact answer.

Here, we iterate towards an answer and when the result no longer changes significantly, the solution has "converged":

$$\varepsilon = \frac{S_{k+1} - S_k}{S_{k+1}}$$

relative change from iteration k to k+1

Note that the sign of the relative error is often not of interest, just its magnitude is important.

Thus, the solution is converged when abs(ε) is small…

Calculation of Error (cont.)

For example, consider an infinite series,

$$S_\infty = \sum_{k=1}^{\infty} T_k = T_1 + T_2 + T_3 \cdots$$

T_k refers to the k^{th} term of the series

The partial sum after k+1 terms, S_{k+1}, minus the partial sum after k terms, S_k, is simply the $(k+1)^{th}$ term in the series,

or

$$T_{k+1} = S_{k+1} - S_k$$

Thus, we have

$$\varepsilon = \frac{T_{k+1}}{S_{k+1}}$$

relative change from iteration k to k+1

If abs(ε) is small, this simply says that the last term added to the running sum was small relative to the total sum…

Calculation of Error (cont.)

As a specific example, consider the following infinite series. Let's estimate S where

$$S = \sum_{k=1}^{\infty} x^{k-1} \quad \text{for} \quad x = 0.8$$

```
S = 0;  k = 1;  x = 0.8;
rerr = 1.0;   tol = 1e-5;
while rerr > tol
  Tk = x^(k-1);
  S = S + Tk;
  rerr = abs(Tk/S);
  k = k+1;
end
S
```

Note that

$$\frac{1}{1-x} = \sum_{k=1}^{\infty} x^{k-1} \quad \text{for} \quad |x| < 1$$

and, for x = 0.8, $S_\infty = 5$

Let's do this in Matlab
(see series_2.m)

Discrete Equations: An Example

Implement the following discrete expression in Matlab:

$$\alpha = x^T A x = \sum_{i=1}^{N} x_i \sum_{j=1}^{N} a_{ij} x_j$$

Case 1: Using Matlab's built-in capability

```
alf = x'*A*x
```

Case 2: Using a discrete representation

```
function alf = qform(A,x)
N = length(x)
alf = 0
for i = 1:N
  sum1 = 0.0
  for j = 1:N
    sum1 = sum1+A(i,j)*x(j)
  end
alf = alf+x(i)*sum1
end
```

Here, two nested loops are required and the result is a scalar

Input and Output Operations

(including Internal & External Documentation)

This is a very broad subject, because input and output operations can involve communication between the user and the program as well as information exchange between programs via data files -- and there are a lot of different options here.

This discussion will only briefly touch on a few key points:

1. Always properly document your programs and analysis results.

   Use internal comments to describe various program sections, the primary variables and their units, and always select meaningful variable names for use within the code…

   Follow these same practices in all the printed and plotted results with the use of *gtext*, *legend*, *title*, *num2str*, *sprintf*, etc., for the plots and *disp* and *fprintf* for printed output…

2. For simple user interaction with Matlab programs, the *input* and *menu* commands are quite useful (for only a few variables).

Input and Output Operations

(including Internal & External Documentation)

3. For more extensive data transfer, there are many options:

   use a separate Matlab script file with the data of interest

   use Matlab's *load* and *save* commands to read/write *.mat files

   read/write space- or comma-delimited ascii files with simple formats with the *dlmread* or *dlmwrite* commands

   write case-specific scripts using the built-in IO routines (*fopen*, *fclose*, *fgetl*, *fscanf*, *textscan*, …) to treat any special cases

   use the built-in import data GUI when appropriate…

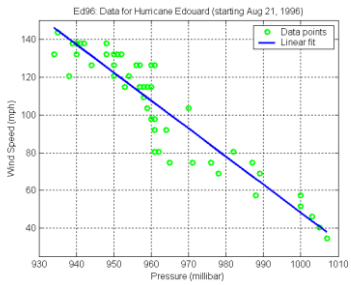
See the formal Lecture Notes for a simulated HW problem that uses the *loadColData.m* file to read/process an ascii data file…

To see many of the available options, type:

```
help iofun
```

CHEN.3170 Applied Engineering Problem Solving
Lesson #3: Programming in Matlab (Sept. 2017)

More Illustrative Examples



Hurricane Edouard (Aug. 21 – Sept. 3, 1996)

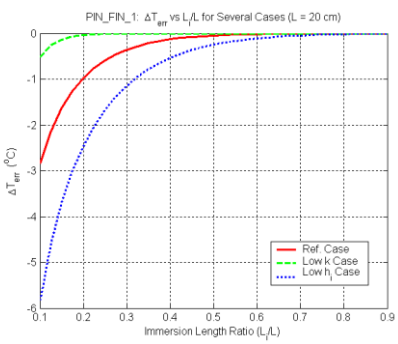
Example that illustrates how to use an m-file as a data file (see ed96.pdf)…

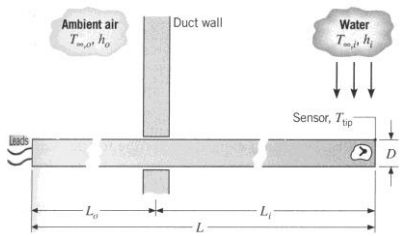
Heat Transfer in a Rectangular Fin

Another problem solving example, where we do the formal model development from base principles, solve the resultant BVP using analytical techniques, and then analyze the results of a parametric study that addresses how the thermal conductivity affects the heat transfer process -- and we also highlight the use of the *fprintf* command in Matlab to prepare some formatted tabular data (see rect1d_fin_1.pdf)...

More Illustrative Examples (cont.)

Measurement Error in a Temperature Probe





Another realistic parametric study that involves the temperature profile and overall heat transfer for a fin with a constant cross-sectional area (see pin_fin_1.pdf) …

Lesson #3 Summary

In this Lesson we have discussed the following topics:

Additional Programming Features

The use of function subprograms for developing well-structured programs

Controlling the flow of a program via conditional tests and looping structures

Implementation of discrete formulas

You should now be much more comfortable with these topics…

Processing input and output data within Matlab (including proper internal documentation)

Developing proper programming logic and problem-solving strategies (via example)…