POLYTECHNIC UNIVERSITY
Department of Financial and Risk Engineering

User-Defined Functions in Matlab

K. Ming Leung

Abstract: Matlab allows user-defined functions written in separate function files. These together with single-line anonymous and inline functions will be discussed.

Directory

Last Revision Date: April 15, 2008 Version 1.0

Table of Contents

1. User-Defined Functions

A user-defined function is a Matlab program that is created by the user, saved as a function file,and then can be used like a built-in function. A function in general has input arguments (or parameters) and output variables (or parameters) that can be scalars, vectors, or matrices of any size. There can be any number of input and output parameters, including zero. Calculations performed inside a function typically make use of the input parameters, and the results of the calculations are transferred out of the function by the output parameters.

2. Writing a Function File

A function file can be written using any text editor (including the Matlab Editor). The file must be in the Matlab Path in order for Matlab to be able to locate the file.

The first executable line in a function file must be the function definition line, which must begin with the keyword `function`. The most general syntax for the function definition line is:

```
function [out1, out2, ...] = functionName(in_1,in2, ...)
```

where "functionName" is the name of the user-defined function, in1, in2, ... are the input parameters, and out1, out2, ... are the output parameters.

The parentheses are needed even if the function has no input parameters:

```
function [out1, out2, ...] = functionName( )
```

If there is only one output parameter, then the square brackets can be omitted:

```
function out = functionName(in1, in2, ...)
```

If there is no output parameter at all, then they are called void functions. The function definition line is written as:

```
function functionName(in1, in2, ...)
```

The first comment line in the function file is referred to as the H1 line. Make sure that the H1 line contains important keywords that adequately describe the function. Because when a user types

```
lookfor aWord
```

on the Command line, Matlab searches for aWord in the H1 lines of all the functions, and if a match is found, the name of the function file as well as the H1 line that contains the match is displayed.

This line is displayed when the user types

```
help functionName
```

in the Command line. This is true for Matlab built-in functions as well as the user-defined functions.

In order for the function file to work, the output arguments **must** be assigned values within the body of the function.

A user-defined function is used in the same way as a built-in function. The function can be called from the Command line, from a script file, or from another function (including itself). A function that calls itself is referred to as a recursive function. To use the function file, the directory where it is located must either be the current directory or be in the search path.

3. Anonymous Functions

An anonymous function is a simple, typical a single line, user-defined function that is defined and written within the computer code (not in a separate file) and is then used in the code. Anonymous functions can be defined in any part of MATLAB (in the Command Window, in script files, and inside regular user-defined functions).

Anonymous functions have been introduced in MATLAB 7. They have several advantages over inline function (to be discussed next). Right now both anonymous and inline functions can be used, but inline function will gradually be phased out.

An anonymous function is created by typing the following statement:

```
functionName = @ (var1,var2,...) expression
```

where 'functionName' is the name of the anonymous function, 'var1', 'var2', etc. are a comma separated list of arguments of the function, and 'expression' is a single mathematical expression involving those variables. The expression can include any built-in or user-defined functions.

The above command creates the anonymous function, and assigns a handle for the function to the variable name on the left of the = sign. Function handles provide means for using the function, and passing it to other functions.

The expression can include predefined variables that are already defined when the anonymous function is defined. For example, if three variables `a`, `b`, and `c` have been assigned values, then they can be used in the expression of the anonymous function:

```
a = 3; b = 4; c = 5;
parabola = @ (x) ( a * x + b ) * x + c
```

It is important to note that MATLAB captures the values of the predefined variables when the anonymous function is defined. This means that if subsequently new values are assigned to the predefined variables, the anonymous function is not changed.

So in the above example, the parabola is always defined so that the three coefficients, `a`, `b`, and `c` are given by 3, 4, and 5, respectively, even though the values of `a`, `b`, and `c` may be altered subsequently.

The anonymous function has to be redefined in order for the new

values of the predefined variables to be used in the expression.

For example,

```
>> FA = @ (x) exp(x.^2)./sqrt(x.^2+5)
```

gives the response

```
FA =
    @(x)exp(x.^2)./sqrt(x.^2+5)
```

Then this anonymous function can be used as follows:

```
>> p = FA(2)
```

which gives the result

```
p = 18.1994
```

and

```
>> Pvec = FA([1 0.5 2]))
```

which gives the result

```
Pvec = 1.1097   0.5604   18.1994
```

4. Inline Functions

Similar to anonymous function, inline function is a simple single-line user-defined function that is defined without creating a separate function file (M-file). Inline functions are gradually being replaced by anonymous functions. We teach it here so that you can understand older MATLAB programs that have inline functions.

An inline function is created with the `inline` command according to the following format:

```
name = inline('mathematical expression typed as a string')
```

A simple example, is:

```
CUBE = inline('X.^3')
```

which calculates the cubic power of an input scalar, vector, or matrix.

For an inline function,

1. the mathematical expression can have one or several independent variables
2. since the arguments are not specified in the above form of the

inline function, MATLAB arranges the arguments in alphabetical order. There is an alternate form that allows the arguments to be specified explicitly.

3. any name except `i` and `j` can be used for the names of the independent variables in the expression
4. the expression cannot include any preassigned variables
5. the expression can include any built-in or user-defined functions
6. once the inline function is defined it can be used by typing its name and a list of values for its arguments in a comma separated list with parentheses.
7. an inline function can be used as an argument in other functions

For example, our previous anonymous function can be defined using the inline function:

```
>> FA = inline('exp(x.^2)./sqrt(x.^2+5)')
FA =
     Inline function:
     FA(x) = exp(x.^2)./sqrt(x.^2+5)
```

```
>> FA(2)
ans  =
18.1994
>> FA([1 0.5 2]))
ans =
    1.1097   0.5604   18.1994
```

An example of an inline function with two arguments (note the assumed ordering of the two arguments):

```
>> HA = inline('y^2+2*x*y+3*x^2')
HA =
     Inline function:
     HA(x,y) = y^2+2*x*y+3*x^2
>> HA(1,2)
ans =
    11
```

There is an alternate form of the inline function that allows the arguments to be specified explicitly:

```
name = inline('mathematical expression typed as a string','ar
```

The argument names as well as their ordering are then specified explicitly.

```
>> HA = inline('y^2+2*x*y+3*x^2','y','x')
HA =
     Inline function:
     HA(y,x) = y^2+2*x*y+3*x^2
>> HA(1,2)
ans =
    17
```

5. Recursive Functions

Functions can be recursive, that is, they can call themselves. Recursion is a powerful tool, but not all computations that are described recursively are best programmed this way.

We will consider the problem of adaptive numerical quadrature as an example of using a recursive function. Numerical quadratures are techniques for numerically computing the integrals of functions. I wrote two such quadratures based on the Gauss-Legendre algorithm,

one using a 10-point and the other one a 12-point formula:

```
gauss10pts(fcn, a, b)
gauss12pts(fcn, a, b)
```

Each of these functions compute numerically the integral of the function `fcn` from a to b. The function `fcn` is either a function handle or a string containing the name of the function file of the integrand function. The more points one uses the more accurate the result is expected to be.

Using these two quadratures, I wrote an adaptive quadrature function that computes the integral of a function from a to b to an accuracy specified by a tolerance, `tol`.

```
function [I,Ct] = adaptiveQuadG2(fcn,a,b,tol,Ct)
% Gander-Gautschi adaptive quadrature with robust
% machine independent termination criterion, &
% avoids arbitrary limits on depth of recursion.
% Using the 10- & 12-point Gauss-Legendre quadrature
% See: Algorithm 8.1 on p.358 Heath
% fcn = (string) name or file-handle of integrand function
% a   = the lower limit.
% b   = the upper limit.
% tol = the tolerance (largest possible error).
% ct  = counter to keep track of the total number of times
%       adaptiveQuadG2 calls itself.
% Total number of function evaluations is 2*(10+12)*ct=44*ct.
% Usage: [I,Ct] = adaptiveQuadG2('invSqrt',0,1,eps,0)
% K. Ming Leung, 08/11/03



% I1 = gauss2pts(fcn, a, b); % optimal choice of points
% I2 = gauss4pts(fcn, a, b); % depends on integrand smoothnes
```

```
I1 = gauss10pts(fcn, a, b);
I2 = gauss12pts(fcn, a, b);
Ct = Ct+1;  % increase counter Ct by 1.

if abs(I2-I1) < tol % works better in Matlab
    I = I2;
else
    m = a + 0.5*(b-a);
    [T1,Ct] = adaptiveQuadG2(fcn,a,m,tol,Ct);
    [T2,Ct] = adaptiveQuadG2(fcn,m,b,tol,Ct);
    I = T1+T2;
end
```