

## SOLVING AND OPTIMIZING IN MATLAB

Note that you can copy code from the pdf and paste into a Matlab editor window to try out the code, or look for the code in Matlab/SuppExamples.

### One-D Solver in Matlab

A single equation can be solved numerically for a single variable using 'fzero'. If you have the optimization toolbox, the solution may be more robust using 'fsolve'. Suppose we wish to solve

$$x^2 + 2x = 1$$

Although the polynomial solution can be found rapidly using the 'roots' function, we will use this function as an example for the use of an iterative solution. The solution can be found iteratively with the single statement.

```
fzero(@(x) (x^2 + 2*x - 1), -3) S-0.1
```

Where -3 is the initial guess and the function returns the solution -2.414. The other solution is found using an initial guess of 1. The @ syntax is used to pass an **anonymous function** to the fzero function (more about anonymous functions later). In this case, it is the complete objective function, and the (x) preceding the function tells matlab that 'x' is the variable to be adjusted. In this case, 'x' is the only variable in the function. To constrain the search interval, a vector with bounds can be used for the second argument, such as [-3 -1]. Note: The fzero function can fail at a discontinuity and the above method does not warn that the returned value is not a solution. It is strongly suggested to be slightly more elegant:

```
[x fval exitflag] = fzero(@(x) (x^2 + 2*x - 1), -3)
```

where we now have a response of the objective function value 'fval', and the exit condition 'exitflag'. The exitflag is 1 on a successful solution. See 'help fzero' otherwise.

For more complicated functions, it is necessary to create a more sophisticated structure, such as shown below. For example, in a binary bubble temperature calculation, the function objective does not fit in a single line because vapor pressures change for both components for a guess on  $T$ , thus we need to put the calculations for checking convergence error within a function statement. Any name can be used for the objective function, in this case 'func'. The variable returned from 'func' is what fzero adjusts to zero. In this case 'obj'. Note that the function name is passed to fzero as a **function handle** using '@func'. Modification of our simple example function is shown here and in Matlab/Examples/fzeroExample.m. An example of a bubble temperature calculation is within Matlab/Chap10/RaoultTx.m

```
function fzeroExample
% this is a 1-D solver example
% the next line sets the command window properties. '0' is the handle.
set(0, 'format', 'shortg'); set(0, 'formatspacing', 'compact');
xguess = [0 1]; %initial interval, try [0 1] and [-3 -1]
[x, fval, exitflag] = fzero(@func, xguess) %call fzero
%successful results if exitflag = 1

function obj = func(x)
    obj = x^2 + 2*x - 1; %definition of objective
end
end
```

### Multi-dimensional Solver in Matlab

Consider the set of equations

$$\begin{aligned}x^2 + 2y &= 10 \\x + y &= 4\end{aligned}$$

## 2 Appendix A Supplement

The solution can be found by transforming the problem to a minimization and using 'fminsearch'. If your Matlab installation has the optimization toolbox, you may wish to use the 'fsolve' routine.

```
function fminsearchExample
% this is a 2-D solver example
set(0,'format','short g'); set(0,'formatspacing','compact');
guess = [1 3]; %initial guess, try [-1 3] and [1 3]
[x,fval,exitflag] = fminsearch(@func, guess) %call fzero
% successful results if exitflag = 1
function obj = func(guess)
    x=guess(1);
    y=guess(2);
    obj1 = x^2 + 2*y - 10;
    obj2 = x + y - 4;
    obj = obj1^2 + obj2^2; %definition of objective
end
end
```

Note that the problem of finding a solution has been transformed into a minimization problem by squaring the objective functions.

### Minimization/Optimization/Function Fitting in Matlab

The 'fminsearch' can be used to optimize and fit constants by creating an objective function. If your installation has the optimization toolbox, the lsqnonlin function may be more robust. An example of fitting is provided in Examples 11.8. Below is a function to fit vapor pressure data using fminsearch to minimize the sum of squares of experimental and calculated data.

```
function fitPsat
% demo of curve fitting
set(0,'format','short g'); set(0,'formatspacing','compact');
%pairs of T(C), Psat(mmHg) for isobutanol
data = [
-9    1;
11.6  5;
21.7  10;
32.4  20;
44.1  40;
51.7  60;
61.5  100;
75.9  200;
91.4  400;
108   760;
];
%extract data
T = data(:,1); Pexpt = log10(data(:,2));
%initial guesses will be in array param
A = 8.5; B = 2000; C = 273;
param = [A B C];
disp('A    B    C')
[param, FVAL, EXITFLAG] = fminsearch(@calcobj,param)
%FVAL and EXITFLAG can be useful if troubleshooting is needed
disp('T    log10(Pexpt)    log10(Pcalc)');
[T Pexpt Pcalc]
disp('log10(Psat) = A - B/(T + C)')
param
plot(1./(T+273.15),Pexpt,'bo');
hold on
plot(1./(T+273.15),Pcalc,'r-'); xlabel('1/T (K^{-1})');
ylabel('log_{10}P(mmHg)');
```

```

function err = calcobj(param)
    A = param(1); B = param(2); C = param(3); %use scalars below
    Pcalc = A - B./(T + C);
    err = sum((Pcalc - Pexpt).^2); % need scalar error for fminsearch
end %calcobj
end

```

## Solving by successive substitution

Occasionally, solutions are needed to complex equations. For example, the equation  $2.5x - \exp(0.75x) = 0$  has two solutions. The solutions can be found by successive substitution. The technique works by rearranging the function in the form  $x = f(x)$ . For the example, there are two possibilities:

$$x = \frac{\exp(0.75x)}{2.5} \quad (\text{A})$$

or

$$x = \frac{\ln(2.5x)}{0.75} \quad (\text{B})$$

Successive substitution works by using  $f(x)$  to generate a new value of  $x$  and then substituting back into  $f(x)$ . The technique will converge only if  $|f'(x)| < 1$  at the value of  $x$  guessed. For the present example, there are two solutions:  $x = \{0.652536, 2.37512\}$ . To solve iteratively for form A, we create a loop that repeats until the value quits changing:

```

set(0,'format','short g'); set(0,'formatspacing','compact');
x = 1;
err = 1;
while (err > 1e-5)
    xold = x;
    x = exp(0.75*x)/2.5;
    err = (xold - x)^2 % watch the results
end
x % echo the answer

```

Note that the iterations are printed to the command window intentionally so that the user can watch to make sure the process is converging. To iterate on the other form, the change is simple:

```

set(0,'format','short g'); set(0,'formatspacing','compact');
x = 3.;
err = 1;
while (err > 1e-5)
    xold = x;
    x = log(2.5*x)/0.75;
    err = (xold - x)^2 % watch the results
end
x % echo the answer

```

## Optimizing Selected Variables From a Set of Function Inputs

Occasionally, optimization of a single variable from a set of function input variables is desired. For example, consider chap08-09/PreosProps.m where a search for 'fzero' will lead to several occurrences, including:

```
[T,obj,exitflag] = fzero(@(T) FindMatch(Tref,Pref,T,P,match),T,options);
```

In this case, FindMatch is passed an entire set of variables, but it is passed as an anonymous function of  $T$ . The conditions for matching are specified in 'match'. This will call FindMatch repeatedly to vary  $T$ , with other variables fixed, until the conditions specified by 'match' are met. For another example, search again for

```
[P,obj,exitflag] = fzero(@(P) FindMatch(Tref, Pref, T, P, match), P, options);
```

where a match is found by adjusting  $P$  with all other variables fixed.

## More on Function Handles and Anonymous Functions

Function handles and anonymous functions are related. They both use the @ character, but in slightly different ways. A function 'handle' is an object used to refer to other objects. As humans, we use 'names', but the computer will use 'handles' to keep track of all kinds of objects. For example, each window or figure has a handle. A plot window in matlab is assigned a handle and the window can be manipulated or objects can be added if the handle is known. Function names can be passed as handles. The code for the textbook usually uses the @ syntax in the fzero statement, which creates the handle.

In Eqn. S-0.1 of this supplement, the fzero was first written using an anonymous function. However, it can also be passed a function handle as in

```
h = @(x) (x^2 + 2*x - 1)
[x fval exitflag] = fzero(h, -3)
```

The first line creates a function handle 'h'. This handle is used in the second line so that Matlab can find the object. To learn more use 'doc handle' or 'docsearch anonymous'. The function fzero is expecting a function handle as the first argument, and can accept an anonymous function or a handle.