Unconstrained Minimization Example

Consider the problem of finding a set of values [x_1, x_2] that solves

$$\underset{x}{\text{minimize}} \quad f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) \qquad \textbf{(2-1)}$$

To solve this two-dimensional problem, write an M-file that returns the function value. Then, invoke the unconstrained minimization routine `fminunc`.

Step 1: Write an M-file objfun.m.

```
function f = objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

Step 2: Invoke one of the unconstrained optimization routines.

```
x0 = [-1,1];    % Starting guess
options = optimset('LargeScale','off');
[x,fval,exitflag,output] = fminunc(@objfun,x0,options)
```

After 40 function evaluations, this produces the solution

```
x =
    0.5000   -1.0000
```

The function at the solution x is returned in `fval`:

```
fval =
  3.6609e-015
```

The `exitflag` tells whether the algorithm converged. `exitflag = 1` means a local minimum was found.

```
exitflag =
     1
```

The `output` structure gives more details about the optimization. For `fminunc`, it includes the number of iterations in `iterations`, the number of function evaluations in `funcCount`, the final step-size in `stepsize`, a measure of first-order optimality (which in this unconstrained case is the infinity norm of the gradient at the solution) in `firstorderopt`, and the type of algorithm used in `algorithm`:

```
output =
       iterations: 8
        funcCount: 66
         stepsize: 1
    firstorderopt: 1.2284e-007
        algorithm: 'medium-scale: Quasi-Newton line search'
          message: [1x85 char]
```

When more than one local minimum exists, the initial guess for the vector [x_1, x_2] affects both the number of function evaluations and the value of the solution point. In the preceding example, `x0` is initialized to `[-1,1]`.

The variable`options` can be passed to `fminunc` to change characteristics of the optimization algorithm, as in

```
x = fminunc(@objfun,x0,options);
```

`options` is a structure that contains values for termination tolerances and algorithm choices. An `options` structure can be created using the `optimset` function:

```
options = optimset('LargeScale','off');
```

In this example, we have turned off the default selection of the large-scale algorithm and so the medium-scale algorithm is used. Other options include controlling the amount of command line display during the optimization iteration, the tolerances for the termination criteria, whether a user-supplied gradient or Jacobian is to be used, and the maximum number of iterations or function evaluations. See`optimset`, the individual optimization functions, and Optimization Options for more options and information.

Examples That Use Standard Algorithms Nonlinear Inequality Constrained Example

Continued on next page...

Nonlinear Inequality Constrained Example

If inequality constraints are added to Equation 2-1, the resulting problem can be solved by the `fmincon` function. For example, find *x* that solves

$$\underset{x}{\text{minimize}}\ f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) \qquad \textbf{(2-2)}$$

subject to the constraints

$$x_1x_2 - x_1 - x_2 \le -1.5$$
$$x_1x_2 \ge -10$$

Because neither of the constraints is linear, you cannot pass the constraints to `fmincon` at the command line. Instead you can create a second M-file, `confun.m`, that returns the value at both constraints at the current `x` in a vector `c`. The constrained optimizer, `fmincon`, is then invoked. Because `fmincon` expects the constraints to be written in the form $c(x) \le 0$, you must rewrite your constraints in the form

$$x_1x_2 - x_1 - x_2 + 1.5 \le 0 \qquad \textbf{(2-3)}$$
$$-x_1x_2 - 10 \le 0$$

Step 1: Write an M-file confun.m for the constraints.

```
function [c, ceq] = confun(x)
% Nonlinear inequality constraints
c = [1.5 + x(1)*x(2) - x(1) - x(2);
     -x(1)*x(2) - 10];
% Nonlinear equality constraints
ceq = [];
```

Step 2: Invoke constrained optimization routine.

```
x0 = [-1,1];     % Make a starting guess at the solution
options = optimset('LargeScale','off');
[x, fval] = ...
fmincon(@objfun,x0,[],[],[],[],[],[],@confun,options)
```

After 38 function calls, the solution `x` produced with function value `fval` is

```
x =
   -9.5474  1.0474
fval =
    0.0236
```

You can evaluate the constraints at the solution by entering

```
[c,ceq] = confun(x)
```

This returns very small numbers close to zero, such as

```
c =
  1.0e-007 *
   -0.9032
    0.9032

ceq =
     []
```

Note that both constraint values are, to within a small tolerance, less than or equal to zero; that is, x satisfies $c(x) \leq 0$.

Unconstrained Minimization Example | Constrained Example with Bounds