fminunc

Find minimum of unconstrained multivariable function

Syntax

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(problem)
[x,fval] = fminunc(...)
[x,fval,exitflag] = fminunc(...)
[x,fval,exitflag,output] = fminunc(...)
[x,fval,exitflag,output,grad] = fminunc(...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Description

fminunc attempts to find a minimum of a scalar function of several variables, starting at an initial estimate. This is generally referred to as unconstrained nonlinear optimization.

x = fminunc(fun,x0) starts at the point x0 and attempts to find a local minimum x of the function described in fun. x0 can be a scalar, vector, or matrix.

x = fminunc(fun,x0,options) minimizes with the optimization options specified in options. Use optimoptions to set these options.

x = fminunc(problem) finds the minimum for problem, where problem is a structure described in Input Arguments.

Create the problem structure by exporting a problem from Optimization app, as described in Exporting Your Work.

[x,fval] = fminunc(...) returns in fval the value of the objective function fun at the solution x.

[x,fval,exitflag] = fminunc(...) returns a value exitflag that describes the exit condition.

[x,fval,exitflag,output] = fminunc(...) returns a structure output that contains information about the optimization.

[x,fval,exitflag,output,grad] = fminunc(...) returns in grad the value of the gradient of fun at the solution x.

[x,fval,exitflag,output,grad,hessian] = fminunc(...) returns in hessian the value of the Hessian of the objective function fun at the solution x. See Hessian.

Examples

Minimize the function $f(x) = 3x_1^2 + 2x_1x_2 + x_2^2$.

Create a file myfun.m:

```
function f = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;    % Cost function
```

Then call fminunc to find a minimum of myfun near [1,1]:

```
x0 = [1,1];
[x,fval] = fminunc(@myfun,x0);
```

After a few iterations, fminunc returns the solution, x, and the value of the function at x, fval:

```
x,fval

x =
  1.0e-006 *
    0.2541   -0.2029

fval =
  1.3173e-013
```

To minimize this function with the gradient provided, modify myfun.m so the gradient is the second output argument:

```
function [f,g] = myfun(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;    % Cost function
if nargout > 1
   g(1) = 6*x(1)+2*x(2);
   g(2) = 2*x(1)+2*x(2);
end
```

Indicate that the gradient value is available by creating optimization options with the GradObj option set to 'on' using optimoptions. Choose the 'trust-region' algorithm, which requires a gradient.

```
options = optimoptions('fminunc','GradObj','on','Algorithm','trust-region');
x0 = [1,1];
[x,fval] = fminunc(@myfun,x0,options);
```

After several iterations `fminunc` returns the solution, `x`, and the value of the function at `x`, `fval`:

```
x,fval

x =
  1.0e-015 *
    0.1110   -0.8882

fval =
  6.2862e-031
```

To minimize the function `f(x) = sin(x) + 3` using an anonymous function

```
f = @(x)sin(x)+3;
x = fminunc(f,4);
```

`fminunc` returns a solution

```
x

x =
    4.7124
```