fminunc

Find minimum of unconstrained multivariable function

Equation

Finds the minimum of a problem specified by

$$\min_x f(x)$$

where x is a vector and $f(x)$ is a function that returns a scalar.

Syntax

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
[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 the structure `options`. Use `optimset` to set these options.

`[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.

Avoiding Global Variables via Anonymous and Nested Functions explains how to parameterize the objective function `fun`, if necessary.

Input Arguments

Function Arguments contains general descriptions of arguments passed into `fminunc`. This section provides function-specific details for `fun` and `options`:

`fun` The function to be minimized. `fun` is a function that accepts a vector `x` and returns a scalar `f`, the objective function evaluated at `x`. The function `fun` can be specified as a function handle for an M-file function

```
x = fminunc(@myfun,x0)
```

where `myfun` is a MATLAB function such as

```
function f = myfun(x)
f = ...            % Compute function value at x
```

`fun` can also be a function handle for an anonymous function.

```
x = fminunc(@(x)norm(x)^2,x0);
```

If the gradient of `fun` can also be computed *and* the `GradObj` option is `'on'`, as set by

```
options = optimset('GradObj','on')
```

then the function `fun` must return, in the second output argument, the gradient value `g`, a vector, at `x`. Note that by checking the value of `nargout` the function can avoid computing `g` when `fun` is called with only one output argument (in the case where the optimization algorithm only needs the value of `f` but not `g`).

```
function [f,g] = myfun(x)
f = ...            % Compute the function value at x
if nargout > 1     % fun called with 2 output arguments
   g = ...         % Compute the gradient evaluated at
end
```

The gradient is the partial derivatives $\partial f / \partial x$ of `f` at the point `x`. That is, the `ith` component of `g` is the partial derivative of `f` with respect to the `ith` component of `x`.

If the Hessian matrix can also be computed *and* the Hessian option is `'on'`, i.e., `options = optimset('Hessian','on')`, then the function `fun` must return the Hessian value `H`, a symmetric matrix, at `x` in a third output argument. Note that by checking the value of `nargout` you can avoid computing `H` when fun is called with only one or two output arguments (in the case where the optimization algorithm only needs the values of `f` and `g` but not `H`).

```
function [f,g,H] = myfun(x)
f = ...      % Compute the objective function value at x
if nargout > 1   % fun called with two output arguments
   g = ...  % Gradient of the function evaluated at x
   if nargout > 2
      H = ...  % Hessian evaluated at x
   end
end
```

The Hessian matrix is the second partial derivatives matrix of `f` at the point `x`. That is, the `(i,j)`th component of `H` is the second partial derivative of `f` with respect to x_i and x_j, $\partial^2 f / \partial x_i \partial x_j$. The Hessian is by definition a symmetric matrix.

`options` Options provides the function-specific details for the `options` values.

Output Arguments

Function Arguments contains general descriptions of arguments returned by `fminunc`. This section provides function-specific details for `exitflag` and `output`:

`exitflag`	Integer identifying the reason the algorithm terminated. The following lists the values of `exitflag` and the corresponding reasons the algorithm terminated.	
	1	Magnitude of gradient smaller than the specified tolerance.
	2	Change in `x` was smaller than the specified tolerance.

		3	Change in the objective function value was less than the specified tolerance.
		0	Number of iterations exceeded `options.MaxIter` or number of function evaluations exceeded `options.FunEvals`.
		-1	Algorithm was terminated by the output function.
		-2	Line search cannot find an acceptable point along the current search direction.
`grad`	Gradient at `x`		
`hessian`	Hessian at `x`		
`output`	Structure containing information about the optimization. The fields of the structure are		
		`iterations`	Number of iterations taken
		`funcCount`	Number of function evaluations
		`algorithm`	Algorithm used
		`cgiterations`	Number of PCG iterations (large-scale algorithm only)
		`stepsize`	Final step size taken (medium-scale algorithm only)

Hessian

`fminunc` computes the output argument `hessian` as follows:

- When using the medium-scale algorithm, the function computes a finite-difference approximation to the Hessian at `x` using
  - The gradient `grad` if you supply it
  - The objective function `fun` if you do not supply the gradient
- When using the large-scale algorithm, the function uses
  - `options.Hessian`, if you supply it, to compute the Hessian at `x`
  - A finite-difference approximation to the Hessian at `x`, if you supply only the gradient

Options

`fminunc` uses these optimization options. Some options apply to all algorithms, some are only relevant when you are using the large-scale algorithm, and others are only relevant when you are using the medium-scale algorithm.You can use `optimset` to set or change the values of these fields in the options structure `options`. See Optimization Options for detailed information.

The `LargeScale` option specifies a *preference* for which algorithm to use. It is only a preference, because certain conditions must be met to use the large-scale algorithm. For `fminunc`, you must provide the gradient (see the preceding description of `fun`) or else use the medium-scale algorithm:

`LargeScale`	Use large-scale algorithm if possible when set to `'on'`. Use medium-scale algorithm when set to `'off'`.

Large-Scale and Medium-Scale Algorithms

These options are used by both the large-scale and medium-scale algorithms:

`DerivativeCheck`	Compare user-supplied derivatives (gradient) to finite-differencing derivatives.
`Diagnostics`	Display diagnostic information about the function to be minimized.
`DiffMaxChange`	Maximum change in variables for finite differencing.
`DiffMinChange`	Minimum change in variables for finite differencing.
`Display`	Level of display. `'off'` displays no output; `'iter'` displays output at each iteration; `'notify'` displays output only if the function does not converge;`'final'` (default) displays just the final output.
`FunValCheck`	Check whether objective function values are valid. `'on'` displays an error when the objective function return a value that is `complex` or `NaN`. `'off'` (the default) displays no error.
`GradObj`	Gradient for the objective function that you define. See the preceding description of `fun` to see how to define the gradient in `fun`.
`MaxFunEvals`	Maximum number of function evaluations allowed.
`MaxIter`	Maximum number of iterations allowed.

`OutputFcn`	Specify one or more user-defined functions that an optimization function calls at each iteration. See Output Function.
`PlotFcns`	Plots various measures of progress while the algorithm executes, select from predefined plots or write your own. Specifying `@optimplotx` plots the current point; `@optimplotfunccount` plots the function count; `@optimplotfval` plots the function value; `@optimplotstepsize` plots the step size; `@optimplotfirstorderopt` plots the first-order of optimality.
`TolFun`	Termination tolerance on the function value.
`TolX`	Termination tolerance on `x`.
`TypicalX`	Typical `x` values.

Large-Scale Algorithm Only

These options are used only by the large-scale algorithm:

`Hessian`	If `'on'`, `fminunc` uses a user-defined Hessian (defined in fun), or Hessian information (when using `HessMult`), for the objective function. If `'off'`, `fminunc` approximates the Hessian using finite differences.

`HessMult`	Function handle for Hessian multiply function. For large-scale structured problems, this function computes the Hessian matrix product `H*Y` without actually forming `H`. The function is of the form
	`W = hmfun(Hinfo,Y,p1,p2,...)`
	where `Hinfo` and possibly the additional parameters `p1,p2,...` contain the matrices used to compute `H*Y`.
	The first argument must be the same as the third argument returned by the objective function fun, for example by
	`[f,g,Hinfo] = fun(x)`

	`Y` is a matrix that has the same number of rows as there are dimensions in the problem. `W = H*Y` although `H` is not formed explicitly. `fminunc` uses `Hinfo` to compute the preconditioner. The optional parameters `p1`, `p2`, `...` can be any additional parameters needed by `hmfun`. See Avoiding Global Variables via Anonymous and Nested Functions for information on how to supply values for the parameters. **Note** `'Hessian'` must be set to `'on'` for `Hinfo` to be passed from `fun` to `hmfun`. See Nonlinear Minimization with a Dense but Structured Hessian and Equality Constraints for an example.
`HessPattern`	Sparsity pattern of the Hessian for finite differencing. If it is not convenient to compute the sparse Hessian matrix `H` in `fun`, the large-scale method in `fminunc` can approximate `H` via sparse finite differences (of the gradient) provided the *sparsity structure* of `H` —i.e., locations of the nonzeros—is supplied as the value for `HessPattern`. In the worst case, if the structure is unknown, you can set `HessPattern` to be a dense matrix and a full finite-difference approximation is computed at each iteration (this is the default). This can be very expensive for large problems, so it is usually worth the effort to determine the sparsity structure.
`MaxPCGIter`	Maximum number of PCG (preconditioned conjugate gradient) iterations (see Algorithms).
`PrecondBandWidth`	Upper bandwidth of preconditioner for PCG. By default, diagonal preconditioning is used (upper bandwidth of 0). For some problems, increasing the bandwidth reduces the number of PCG iterations. Setting `PrecondBandWidth` to `'Inf'` uses a direct factorization (Cholesky) rather than the conjugate gradients (CG). The direct factorization is computationally more expensive than CG, but produces a better quality step towards the solution.
`TolPCG`	Termination tolerance on the PCG iteration.

Medium-Scale Algorithm Only

These options are used only by the medium-scale algorithm:

`HessUpdate`	Method for choosing the search direction in the Quasi-Newton algorithm. The choices are • `'bfgs'` • `'dfp'` • `'steepdesc'` See Hessian Update for a description of these methods.
`InitialHessMatrix`	Initial quasi-Newton matrix. This option is only available if you set `InitialHessType` to `'user-supplied'`. In that case, you can set `InitialHessMatrix` to one of the following: • `scalar` — the initial matrix is the scalar times the identity • `vector` — the initial matrix is a diagonal matrix with the entries of the vector on the diagonal.
`InitialHessType`	Initial quasi-Newton matrix type. The options are • `'identity'` • `'scaled-identity'` • `'user-supplied'`

Examples

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

To use an M-file, 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 couple of iterations, the solution, `x`, and the value of the function at `x`, fval, are returned.

```
x =

  1.0e-006 *

    0.2541   -0.2029


fval =

  1.3173e-013
```

To minimize this function with the gradient provided, modify the M-file 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
```

and indicate that the gradient value is available by creating an optimization options structure with the GradObj option set to 'on' using optimset.

```
options = optimset('GradObj','on');
x0 = [1,1];
[x,fval] = fminunc(@myfun,x0,options)
```

After several iterations the solution, x, and fval, the value of the function at x, are returned.

```
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)
```

which returns a solution

```
x =
    4.7124
```

Notes

`fminunc` is not the preferred choice for solving problems that are sums of squares, that is, of the form

$$\min_x (f(x)) = f_1(x)^2 + f_2(x)^2 + f_3(x)^2 + \ldots + f_m(x)^2$$

Instead use the `lsqnonlin` function, which has been optimized for problems of this form.

To use the large-scale method, you must provide the gradient in `fun` (and set the `GradObj` option to `'on'` using `optimset`). A warning is given if no gradient is provided and the `LargeScale` option is not `'off'`.

Algorithms

Large-Scale Optimization

By default `fminunc` chooses the large-scale algorithm if you supplies the gradient in `fun` (and the `GradObj` option is set to `'on'` using `optimset`). This algorithm is a subspace trust region method and is based on the interior-reflective Newton method described in [2] and [3]. Each iteration involves the approximate solution of a large linear system using the method of preconditioned conjugate gradients (PCG). See Trust-Region Methods for Nonlinear Minimization and Preconditioned Conjugate Gradients.

Medium-Scale Optimization

`fminunc`, with the `LargeScale` option set to `'off'` with `optimset`, uses the BFGS Quasi-Newton method with a cubic line search procedure. This quasi-Newton method uses the BFGS ([1],[5],[8], and [9]) formula for updating the approximation of the Hessian matrix. You can select the DFP ([4],[6], and [7]) formula, which approximates the inverse Hessian matrix, by setting the `HessUpdate` option to `'dfp'` (and the `LargeScale` option to `'off'`). You can select a steepest descent method by setting `HessUpdate` to `'steepdesc'` (and `LargeScale` to `'off'`), although this is not recommended.

Limitations

The function to be minimized must be continuous. `fminunc` might only give local solutions.

`fminunc` only minimizes over the real numbers, that is, *x* must only consist of real numbers and *f*(*x*) must only return real numbers. When *x* has complex variables, they must be split into real and imaginary parts.

Large-Scale Optimization

To use the large-scale algorithm, you must supply the gradient in `fun` (and `GradObj` must be set `'on'` in `options`). See Large-Scale Problem Coverage and Requirements for more information on what problem formulations are covered and what information must be provided.

References

[1] Broyden, C.G., "The Convergence of a Class of Double-Rank Minimization Algorithms," *Journal Inst. Math. Applic.*, Vol. 6, pp. 76-90, 1970.

[2] Coleman, T.F. and Y. Li, "An Interior, Trust Region Approach for Nonlinear Minimization Subject to Bounds," *SIAM Journal on Optimization*, Vol. 6, pp. 418-445, 1996.

[3] Coleman, T.F. and Y. Li, "On the Convergence of Reflective Newton Methods for Large-Scale Nonlinear Minimization Subject to Bounds," *Mathematical Programming*, Vol. 67, Number 2, pp. 189-224, 1994.

[4] Davidon, W.C., "Variable Metric Method for Minimization," *A.E.C. Research and Development Report*, ANL-5990, 1959.

[5] Fletcher, R., "A New Approach to Variable Metric Algorithms," *Computer Journal*, Vol. 13, pp. 317-322, 1970.

[6] Fletcher, R., "Practical Methods of Optimization," Vol. 1, *Unconstrained Optimization*, John Wiley and Sons, 1980.

[7] Fletcher, R. and M.J.D. Powell, "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, Vol. 6, pp. 163-168, 1963.

[8] Goldfarb, D., "A Family of Variable Metric Updates Derived by Variational Means," *Mathematics of Computing*, Vol. 24, pp. 23-26, 1970.

[9] Shanno, D.F., "Conditioning of Quasi-Newton Methods for Function Minimization," *Mathematics of Computing*, Vol. 24, pp. 647-656, 1970.

See Also

`@` (`function_handle`), `fminsearch`, `optimset`, `optimtool`, anonymous functions