本页的翻译已过时。点击此处可查看最新英文版本。

fsolve

对非线性方程组求解

语法

```
x = fsolve(fun,x0)
x = fsolve(fun,x0,options)
x = fsolve(problem)
[x,fval] = fsolve( ___ )
[x,fval,exitflag,output] = fsolve( ___ )
[x,fval,exitflag,output,jacobian] = fsolve( ___ )
```

说明

非线性方程组求解器

对于下式指定的问题

F(x) = 0

求解其中的 x，其中 F(x) 是返回向量值的函数。

x 是向量或矩阵；请参阅矩阵参数。

`x = fsolve(fun,x0)` 从 `x0` 开始，尝试求解方程 `fun(x) = 0`（由零组成的数组）。	示例
`x = fsolve(fun,x0,options)` 使用 `options` 中指定的优化选项求解方程。使用 `optimoptions` 可设置这些选项。	示例
`x = fsolve(problem)` 求解 `problem`，它是 `problem` 中所述的一个结构体。	示例
`[x,fval] = fsolve( ___ )` 对上述任何语法，返回目标函数 `fun` 在解 `x` 处的值。	示例
`[x,fval,exitflag,output] = fsolve( ___ )` 还返回描述 `fsolve` 的退出条件的值 `exitflag`，以及提供优化过程信息的结构体 `output`。	示例
`[x,fval,exitflag,output,jacobian] = fsolve( ___ )` 返回 `fun` 在解 `x` 处的 Jacobian 矩阵。	

示例

全部折叠

二维非线性方程组的求解

此示例说明如何求解包含两个变量的两个非线性方程。这些方程包括

Copy Command

$$e^{-e^{-(x_1+x_2)}} = x_2\left(1+x_1^2\right)$$
$$x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) = \tfrac{1}{2}.$$

将方程转换为 $F(x)=0$ 形式。

$$e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) = 0$$
$$x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) - \tfrac{1}{2} = 0.$$

编写一个函数来计算这两个方程的左侧。

```
function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

将此代码保存为 MATLAB® 路径上名为 `root2d.m` 的文件。

从 `[0,0]` 点开始求解方程组。

```
fun = @root2d;
x0 = [0,0];
x = fsolve(fun,x0)
```

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.


x =

    0.3532    0.6061
```

使用非默认选项的求解

深入了解非线性方程组的求解过程。

Copy Command

将选项设置为无显示和显示一阶最优性的绘图函数，该函数应在算法迭代时收敛于 0。

```
options = optimoptions('fsolve','Display','none','PlotFcn',@optimplotfirstorderopt);
```

非线性方程组中的方程是

$$\begin{aligned} e^{-e^{-(x_1+x_2)}} &= x_2\left(1+x_1^2\right) \\ x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) &= \tfrac{1}{2}. \end{aligned}$$

将方程转换为 $F(x)=\mathbf{0}$ 形式。

$$\begin{aligned} e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) &= 0 \\ x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) - \tfrac{1}{2} &= 0. \end{aligned}$$

编写一个函数来计算这两个方程的左侧。

```
function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

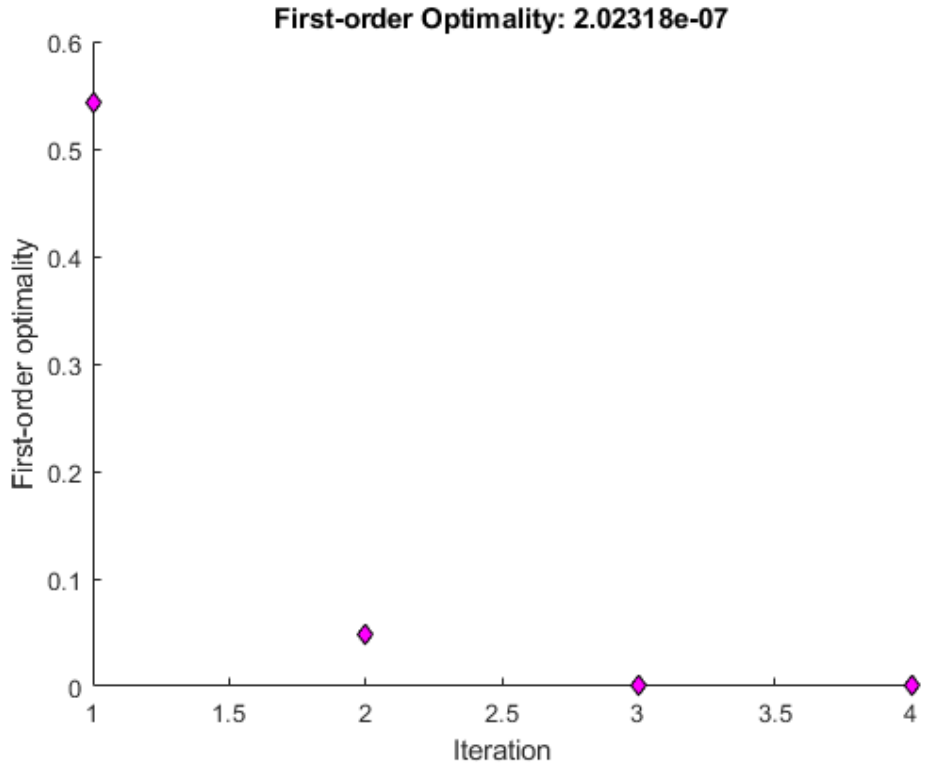
将此代码保存为 MATLAB® 路径上名为 `root2d.m` 的文件。

从点 `[0,0]` 开始求解非线性方程组，并观察求解过程。

```
fun = @root2d;
x0 = [0,0];
x = fsolve(fun,x0,options)
```

```
x =

    0.3532    0.6061
```



求解问题结构体

为 fsolve 创建问题结构体并求解问题。

Copy Command

求解与使用非默认选项的求解中相同的问题，但使用问题结构体来表示问题。

设置问题的相关选项，不显示迭代输出，使用绘图函数显示一阶最优性，一阶最优性应随着算法迭代而收敛于 0。

```
problem.options = optimoptions('fsolve','Display','none','PlotFcn',@optimplotfirstorderopt);
```

非线性方程组中的方程是

$$\begin{aligned} e^{-e^{-(x_1+x_2)}} &= x_2\left(1+x_1^2\right) \\ x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) &= \tfrac{1}{2}. \end{aligned}$$

将方程转换为 $F(x)=\mathbf{0}$ 形式。

$$\begin{aligned} e^{-e^{-(x_1+x_2)}} - x_2\left(1+x_1^2\right) &= 0 \\ x_1\cos\left(x_2\right)+x_2\sin\left(x_1\right) - \tfrac{1}{2} &= 0. \end{aligned}$$

编写一个函数来计算这两个方程的左侧。

```
function F = root2d(x)

F(1) = exp(-exp(-(x(1)+x(2)))) - x(2)*(1+x(1)^2);
F(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
```

将此代码保存为 MATLAB® 路径上名为 root2d.m 的文件。

在问题结构体中创建其余字段。

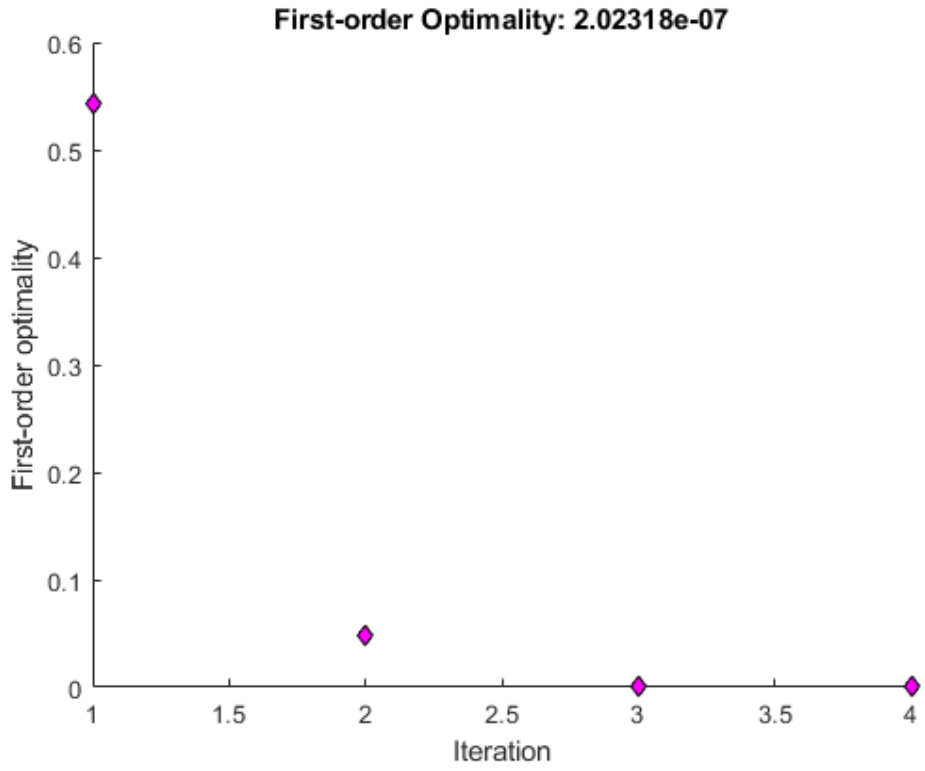
```
problem.objective = @root2d;
problem.x0 = [0,0];
problem.solver = 'fsolve';
```

求解。

```
x = fsolve(problem)
```

```
x =

    0.3532    0.6061
```



非线性方程组的求解过程

此示例返回迭代输出，展示了一个包含两个方程和两个未知数的方程组的求解过程

Copy Command

$$2x_1 - x_2 = e^{-x_1}$$
$$-x_1 + 2x_2 = e^{-x_2}.$$

以 $F(x) = 0$ 形式重写方程：

$$2x_1 - x_2 - e^{-x_1} = 0$$
$$-x_1 + 2x_2 - e^{-x_2} = 0.$$

从 `x0 = [-5 -5]` 开始搜索解。

首先，编写一个函数用来计算 `F`（方程在 `x` 处的值）。

```
F = @(x) [2*x(1) - x(2) - exp(-x(1));
         -x(1) + 2*x(2) - exp(-x(2))];
```

创建初始点 `x0`。

```
x0 = [-5;-5];
```

设置选项以返回迭代输出。

```
options = optimoptions('fsolve','Display','iter');
```

求解方程。

```
[x,fval] = fsolve(F,x0,options)
```

```
                                         Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          3         47071.2                     2.29e+04              1
     1          6         12003.4             1       5.75e+03              1
     2          9         3147.02             1       1.47e+03              1
     3         12         854.452             1            388              1
     4         15         239.527             1            107              1
     5         18         67.0412             1           30.8              1
     6         21         16.7042             1           9.05              1
     7         24         2.42788             1           2.26              1
     8         27        0.032658      0.759511          0.206            2.5
     9         30     7.03149e-06      0.111927        0.00294            2.5
    10         33     3.29525e-13     0.00169132       6.36e-07            2.5

 Equation solved.

 fsolve completed because the vector of function values is near zero
 as measured by the value of the function tolerance, and
 the problem appears regular as measured by the gradient.
 x = 2×1

     0.5671
     0.5671

 fval = 2×1
 10^-6 ×

    -0.4059
    -0.4059
```

迭代输出显示 `f(x)`，这是函数 `F(x)` 的范数的平方。随着迭代的进行，该值减小到接近于零。随着迭代的进行，一阶最优性度量同样减小到接近于零。这些条目显示迭代收敛于一个解。有关其他条目的含义，请参阅迭代输出。

`fval` 输出给出函数值 `F(x)`，该值在解处应为零（在 `FunctionTolerance` 容差内）。

检查矩阵方程解

找到满足下式的矩阵 X

$$X*X*X=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},$$

Copy Command

初始点为 `x0 = [1,1;1,1]`。创建匿名函数来计算矩阵方程并创建点 `x0`。

```
fun = @(x)x*x*x - [1,2;3,4];
x0 = ones(2);
```

将选项设置为不显示。

```
options = optimoptions('fsolve','Display','off');
```

检查 fsolve 输出，了解求解过程和解的质量。

```
[x,fval,exitflag,output] = fsolve(fun,x0,options)
```

```
x = 2×2

   -0.1291    0.8602
    1.2903    1.1612

fval = 2×2
10^-9 ×

   -0.1618    0.0778
    0.1160   -0.0474

exitflag = 1
output = struct with fields:
       iterations: 6
        funcCount: 35
        algorithm: 'trust-region-dogleg'
    firstorderopt: 2.4095e-10
          message: '...'
```

退出标志值 1 表示解是可靠的。要手动验证解的可靠性，请计算残差（fval 的平方和）并观察它与零的接近程度。

```
sum(sum(fval.*fval))
```

```
ans = 4.7957e-20
```

这里残差很小，证实 x 是一个解。

您可以在 output 结构体中看到 fsolve 执行了多少次迭代和函数计算才求得解。

输入参数

全部折叠

fun - 要求解的非线性方程

函数句柄 | 函数名称

要求解的非线性方程，指定为函数句柄或函数名称。fun 是函数，它接受向量 x 并返回向量 F，即在 x 处计算的非线性方程。对于 F 的所有分量，要求解的方程是 F = 0。对于函数文件，函数 fun 可以指定为函数句柄

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

其中 myfun 是一个 MATLAB® 函数，例如

```
function F = myfun(x)
F = ...            % Compute function values at x
```

fun 也可以是匿名函数的函数句柄。

```
x = fsolve(@(x)sin(x.*x),x0);
```

如果 x 和 F 的用户定义值是数组，则使用线性索引将它们转换为向量（请参阅数组索引）。

如果 Jacobian 矩阵也可以计算并且 'SpecifyObjectiveGradient' 选项为 true，设置如下

```
options = optimoptions('fsolve','SpecifyObjectiveGradient',true)
```

函数 fun 必须在第二个输出参数中返回 x 处的 Jacobian 值 J，它是一个矩阵。

如果 fun 返回由 m 个分量组成的向量（矩阵）并且 x 的长度为 n，其中 n 是 x0 的长度，则 Jacobian 值 J 是 m×n 矩阵，其中 J(i,j) 是 F(i) 关于 x(j) 的偏导数。（Jacobian 值 J 是 F 的梯度的转置。）

示例： `fun = @(x)x*x*x-[1,2;3,4]`

数据类型： `char` | `function_handle` | `string`

x0 - 初始点

实数向量 | 实数数组

初始点，指定为实数向量或实数数组。fsolve 使用 x0 中的元素数及其大小来确定 fun 接受的变量的数目和大小。

示例： `x0 = [1,2,3,4]`

数据类型： `double`

options - 优化选项

optimoptions 的输出 | optimset 返回的结构体

优化选项，指定为 optimoptions 的输出或 optimset 等返回的结构体。

一些选项适用于所有算法，其他选项则与特定算法相关。有关详细信息，请参阅优化选项参考。

optimoptions 显示中缺少某些选项。这些选项在下表中以斜体显示。有关详细信息，请参阅View Options。

所有算法	
Algorithm	在 'trust-region-dogleg'（默认值）、'trust-region' 和 'levenberg-marquardt' 之间进行选择。 Algorithm 选项指定算法使用的预设项。它只是一个预设项，因为对于信赖域算法，非线性方程组不能为欠定；也就是说，方程的数目（fun 返回的 F 的元素数）必须至少与 x 的长度相同。同样，对于信赖域 dogleg 算法，方程的数目必须与 x 的长度相同。fsolve 在所选算法不可用时，使用 Levenberg-Marquardt 算法。有关选择算法的详细信息，请参阅选择算法。 对于部分算法选项，要使用 optimset 而不是 optimoptions 进行设置，请指定如下： • Algorithm - 将算法设置为 'trust-region-reflective' 而不是 'trust-region'。 • InitDamping - 通过将 Algorithm 设置为形如 {'levenberg-marquardt',.005} 的元胞数组，来设置初始 Levenberg-Marquardt 参数 λ。
CheckGradients	将用户提供的导数（目标或约束的梯度）与有限差分导数进行比较。选择项是 true 或默认值 false。 对于 optimset，名称为 DerivativeCheck，值为 'on' 或 'off'。请参阅当前选项名称和旧选项名称。
Diagnostics	显示关于要最小化或求解的函数的诊断信息。选择项是 'on' 或默认值 'off'。
DiffMaxChange	有限差分梯度变量的最大变化（正标量）。默认值为 Inf。
DiffMinChange	有限差分梯度变量的最小变化（正标量）。默认值为 0。

`Display`	显示级别（请参阅迭代输出）： • `'off'` 或 `'none'` 不显示输出。 • `'iter'` 显示每次迭代的输出，并给出默认退出消息。 • `'iter-detailed'` 显示每次迭代的输出，并给出带有技术细节的退出消息。 • `'final'`（默认值）仅显示最终输出，并给出默认退出消息。 • `'final-detailed'` 仅显示最终输出，并给出带有技术细节的退出消息。

`FiniteDifferenceStepSize`	有限差分的标量或向量步长大小因子。当您将 `FiniteDifferenceStepSize` 设置为向量 `v` 时，前向有限差分 `delta` 是
	`delta = v.*sign’(x).*max(abs(x),TypicalX);`
	其中 `sign’(x) = sign(x)`（例外是 `sign’(0) = 1`）。中心有限差分是
	`delta = v.*max(abs(x),TypicalX);`
	标量 `FiniteDifferenceStepSize` 扩展为向量。对于正向有限差分，默认值为 `sqrt(eps)`；对于中心有限差分，默认值为 `eps^(1/3)`。
	对于 `optimset`，名称是 `FinDiffRelStep`。请参阅当前选项名称和旧选项名称。

`FiniteDifferenceType`	用于估计梯度的有限差分是 `'forward'`（默认值）或 `'central'`（中心化）。`'central'` 需要两倍的函数计算次数，但应更准确。 当同时估计这两种类型的有限差分时，该算法小心地遵守边界。因此，例如，为了避免在边界之外的某个点进行计算，它可能采取一个后向差分，而不是前向差分。 对于 `optimset`，名称是 `FinDiffType`。请参阅当前选项名称和旧选项名称。
`FunctionTolerance`	关于函数值的终止容差，为正标量。默认值为 `1e-6`。请参阅容差和停止条件。 对于 `optimset`，名称是 `TolFun`。请参阅当前选项名称和旧选项名称。
`FunValCheck`	检查目标函数值是否有效。如果选择 `'on'`，则在目标函数返回的值是 `complex`、`Inf` 或 `NaN` 时，将显示错误。默认值 `'off'` 不会显示错误。
`MaxFunctionEvaluations`	允许的函数计算的最大次数，为正整数。默认值为 `100*numberOfVariables`。请参阅容差和停止条件和迭代和函数计算次数。 对于 `optimset`，名称是 `MaxFunEvals`。请参阅当前选项名称和旧选项名称。
`MaxIterations`	允许的迭代最大次数，为正整数。默认值为 `400`。请参阅容差和停止条件和迭代和函数计算次数。 对于 `optimset`，名称是 `MaxIter`。请参阅当前选项名称和旧选项名称。
`OptimalityTolerance`	一阶最优性的终止容差（正标量）。默认值为 `1e-6`。请参阅一阶最优性度量。 `'levenberg-marquardt'` 算法在内部使用 `1e-4` 乘以 `FunctionTolerance` 作为最优性容差（停止条件），而不使用 `OptimalityTolerance`。
`OutputFcn`	指定优化函数在每次迭代中调用的一个或多个用户定义的函数。传递函数句柄或函数句柄的元胞数组。默认值是“无”（`[]`）。请参阅Output Function and Plot Function Syntax。

`PlotFcn`	对算法执行过程中的各种进度测量值绘图，可以选择预定义的绘图，也可以自行编写绘图函数。传递内置绘图函数名称、函数句柄或由内置绘图函数名称或函数句柄组成的元胞数组。对于自定义绘图函数，传递函数句柄。默认值是“无”（`[]`）： • `'optimplotx'` 绘制当前点。 • `'optimplotfunccount'` 绘制函数计数。 • `'optimplotfval'` 绘制函数值。 • `'optimplotstepsize'` 绘制步长大小。 • `'optimplotfirstorderopt'` 绘制一阶最优性度量。 自定义绘图函数使用与输出函数相同的语法。请参阅Output Functions for Optimization Toolbox™和Output Function and Plot Function Syntax。 对于 `optimset`，名称是 `PlotFcns`。请参阅当前选项名称和旧选项名称。
`SpecifyObjectiveGradient`	如果为 `true`，则对于目标函数，`fsolve` 使用用户定义的 Jacobian 矩阵（在 `fun` 中定义）或 Jacobian 信息（使用 `JacobianMultiplyFcn` 时）。如果为 `false`（默认值），`fsolve` 使用有限差分逼近 Jacobian 矩阵。 对于 `optimset`，名称为 `Jacobian`，值为 `'on'` 或 `'off'`。请参阅当前选项名称和旧选项名称。
`StepTolerance`	关于正标量 `x` 的终止容差。默认值为 `1e-6`。请参阅容差和停止条件。 对于 `optimset`，名称是 `TolX`。请参阅当前选项名称和旧选项名称。
`TypicalX`	典型的 `x` 值。`TypicalX` 中的元素数等于 `x0`（即起点）中的元素数。默认值为 `ones(numberofvariables,1)`。`fsolve` 使用 `TypicalX` 缩放有限差分来进行梯度估计。 `trust-region-dogleg` 算法使用 `TypicalX` 作为缩放矩阵的对角项。
`UseParallel`	此选项为 `true` 时，`fsolve` 以并行方式估计梯度。要禁用，请将其设置为默认值 `false`。请参阅并行计算。
信赖域算法	

`JacobianMultiplyFcn`	Jacobian 矩阵乘法函数，指定为函数句柄。对于大规模结构问题，此函数计算 Jacobian 矩阵乘积 `J*Y`、`J'*Y` 或 `J'*(J*Y)`，而并不实际构造 `J`。函数的形式是
	`W = jmfun(Jinfo,Y,flag)`
	其中 `Jinfo` 包含用于计算 `J*Y`（或 `J'*Y`、`J'*(J*Y)`）的矩阵。第一个参数 `Jinfo` 必须与目标函数 `fun` 返回的第二个参数相同，例如下式
	`[F,Jinfo] = fun(x)`
	`Y` 是矩阵，其行数与问题中的维数相同。`flag` 确定要计算的乘积：
	• 如果 `flag == 0`，则 `W = J'*(J*Y)`。
	• 如果 `flag > 0`，则 `W = J*Y`。
	• 如果 `flag < 0`，则 `W = J'*Y`。
	以上任一情形都没有显式构造 `J`。`fsolve` 使用 `Jinfo` 计算预条件子。有关如何为 `jmfun` 所需的额外参数提供值的信息，请参阅传递额外参数。
	注意
	`'SpecifyObjectiveGradient'` 必须设置为 `true`，`fsolve` 才能将 `Jinfo` 从 `fun` 传递到 `jmfun`。
	请参阅Minimization with Dense Structured Hessian, Linear Equalities了解类似示例。
	对于 `optimset`，名称是 `JacobMult`。请参阅当前选项名称和旧选项名称。

JacobPattern	用于有限差分的 Jacobian 矩阵稀疏模式。当 `fun(i)` 依赖 `x(j)` 时，设置 `JacobPattern(i,j) = 1`。否则，设置 `JacobPattern(i,j) = 0`。换句话说，如果存在 ∂`fun(i)`/∂`x(j)` ≠ 0，则 `JacobPattern(i,j) = 1`。 如果不方便计算 `fun` 的 Jacobian 矩阵 `J`，但您可以（例如通过分析）确定 `fun(i)` 何时依赖 `x(j)`，请使用 `JacobPattern`。如果您提供 `JacobPattern`，`fsolve` 可以通过稀疏有限差分逼近 `J`。 在最坏的情形下，如果结构未知，不要设置 `JacobPattern`。默认行为是将 `JacobPattern` 视为由 1 组成的稠密矩阵。然后，`fsolve` 在每次迭代中计算满有限差分逼近。对于大型问题，这种计算可能成本非常高昂，因此通常最好确定稀疏结构。
MaxPCGIter	PCG（预条件共轭梯度）迭代的最大次数，正标量。默认值为 `max(1,floor(numberOfVariables/2))`。有关详细信息，请参阅方程求解算法。
PrecondBandWidth	PCG 的预条件子上带宽，非负整数。`PrecondBandWidth` 的默认值是 `Inf`，这意味着使用直接分解 (Cholesky)，而不是共轭梯度 (CG)。直接分解的计算成本较 CG 高，但所得的求解步质量更好。将 `PrecondBandWidth` 设置为 `0` 将使用对角预条件（上带宽为 0）。对于某些问题，中间带宽会减少 PCG 迭代的次数。
`SubproblemAlgorithm`	确定迭代步的计算方式。相比 `'cg'`，默认值 `'factorization'` 采用的迭代步较慢，但更准确。请参阅信赖域算法。
TolPCG	PCG 迭代的终止容差，正标量。默认值为 `0.1`。
Levenberg-Marquardt 算法	
InitDamping	Levenberg-Marquardt 参数的初始值，正标量。默认值是 `1e-2`。有关详细信息，请参阅Levenberg-Marquardt 方法。
ScaleProblem	`'jacobian'` 有时可以改进缩放不良的问题的收敛性。默认值为 `'none'`。

示例： `options = optimoptions('fsolve','FiniteDifferenceType','central')`

`problem` - 问题结构体

结构体

问题结构体，指定为含有以下字段的结构体：

字段名称	条目
`objective`	目标函数
`x0`	x 的初始点
`solver`	`'fsolve'`
`options`	用 `optimoptions` 创建的选项

数据类型： `struct`

输出参数

全部折叠

`x` - 解

实数向量 | 实数数组

解，以实数向量或实数数组形式返回。x 的大小与 `x0` 的大小相同。通常情况下，当 `exitflag` 为正时，x 是该问题的局部解。有关解质量的信息，请参阅When the Solver Succeeds。

`fval` - 解处的目标函数值

实数向量

解处的目标函数值，以实数向量形式返回。通常，`fval = fun(x)`。

`exitflag` - `fsolve` 停止的原因

整数

`fsolve` 停止的原因，以整数形式返回。

`1`	方程已解。一阶最优性很小。
`2`	方程已解。x 中的变化小于指定容差，或 x 处的 Jacobian 矩阵未定义。
`3`	方程已解。残差的变化小于指定容差。
`4`	方程已解。搜索方向的模小于指定容差。
`0`	迭代次数超出 `options.MaxIterations` 或函数计算次数超过 `options.MaxFunctionEvaluations`。
`-1`	输出函数或绘图函数使算法停止。
`-2`	方程未得解。退出消息可能包含详细信息。
`-3`	方程未得解。信赖域半径变得太小（`trust-region-dogleg` 算法）。

`output` - 有关优化过程的信息

结构体

有关优化过程的信息，以包含下列字段的结构体形式返回：

`iterations`	执行的迭代次数
`funcCount`	函数计算次数
`algorithm`	使用的优化算法
`cgiterations`	PCG 迭代总数（仅适用于 `'trust-region'` 算法）
`stepsize`	`x` 的最终位移（不适用于 `'trust-region-dogleg'`）
`firstorderopt`	一阶最优性的度量
`message`	退出消息

`jacobian` - 解处的 Jacobian 矩阵

实矩阵

解处的 Jacobian 矩阵，以实矩阵形式返回。`jacobian(i,j)` 是 `fun(i)` 关于解 `x` 处的 `x(j)` 的偏导数。

局限性

- 要求解的函数必须为连续的。
- 成功求解后，`fsolve` 只给出一个根。
- 仅当方程组为方阵（即方程的数目等于未知数的数目）时，才使用默认的信赖域 dogleg 方法。如使用 Levenberg-Marquardt 方法，方程组不必为方阵。

提示

- 对于大型问题，即变量数以千计、甚至更多的问题，将 `Algorithm` 选项设置为 `'trust-region'` 并将 `SubproblemAlgorithm` 选项设置为 `'cg'` 以节省内存（同时也可能节省时间）。

算法

Levenberg-Marquardt 和信赖域方法基于非线性最小二乘算法，该算法在 `lsqnonlin` 中也有使用。如果方程组不能包含零，请使用这些方法之一。该算法仍返回残差很小的点。然而，如果方程组的 Jacobian 矩阵是奇异的，则算法收敛到的点可能并非方程组的解（请参阅局限性）。

- 默认情况下，`fsolve` 选择信赖域 dogleg 算法。该算法是 [8] 中所述的 Powell dogleg 方法的变体。它在本质上类似于在 [7] 中实现的算法。请参阅信赖域 dogleg 算法。
- 信赖域算法是一种子空间信赖域方法，基于 [1] 和 [2] 中所述的内部反射牛顿法。每次迭代都涉及使用预条件共轭梯度法 (PCG) 来近似求解大型线性方程组。请参阅信赖域算法。
- 参考文献 [4]、[5] 和 [6] 中描述了 Levenberg-Marquardt 方法。请参阅 Levenberg-Marquardt 方法。

替代功能

App

优化实时编辑器任务为 `fsolve` 提供可视化界面。

参考

[1] 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.

[2] 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.

[3] Dennis, J. E. Jr., "Nonlinear Least-Squares," State of the Art in Numerical Analysis, ed. D. Jacobs, Academic Press, pp. 269-312.

[4] Levenberg, K., "A Method for the Solution of Certain Problems in Least-Squares," Quarterly Applied Mathematics 2, pp. 164-168, 1944.

[5] Marquardt, D., "An Algorithm for Least-squares Estimation of Nonlinear Parameters," SIAM Journal Applied Mathematics, Vol. 11, pp. 431-441, 1963.

[6] Moré, J. J., "The Levenberg-Marquardt Algorithm: Implementation and Theory," Numerical Analysis, ed. G. A. Watson, Lecture Notes in Mathematics 630, Springer Verlag, pp. 105-116, 1977.

[7] Moré, J. J., B. S. Garbow, and K. E. Hillstrom, User Guide for MINPACK 1, Argonne National Laboratory, Rept. ANL-80-74, 1980.

[8] Powell, M. J. D., "A Fortran Subroutine for Solving Systems of Nonlinear Algebraic Equations," Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Ch.7, 1970.

扩展功能

- **C/C++ 代码生成**
  使用 MATLAB® Coder™ 生成 C 代码和 C++ 代码。
- **自动并行支持**
  通过使用 Parallel Computing Toolbox™ 自动运行并行计算来加快代码执行。

另请参阅

`fzero` | `lsqcurvefit` | `lsqnonlin` | `optimoptions` | 优化

主题

Solve Nonlinear System Without and Including Jacobian
Large Sparse System of Nonlinear Equations with Jacobian
Large System of Nonlinear Equations with Jacobian Sparsity Pattern
带约束的非线性方程组
基于求解器的优化问题设置
方程求解算法

在 R2006a 之前推出