第 3 章 无约束最优化

本 章 内 容

3.1 无约束最优化问题简介

无约束最优化问题的数学模型；无约束最优化问题的解析解方法；无约束最优化问题的图解法；局部最优解与全局最优解；数值求解算法的 MATLAB 实现。

3.2 无约束最优化问题的 MATLAB 直接求解

直接求解方法；最优化控制选项；最优搜索中间过程的图形显示；附加参数的传递；最优化问题的结构体描述；梯度信息与求解精度；基于问题的描述方法；离散点最优化问题的求解；最优化问题的并行求解。

3.3 全局最优解的尝试

全局最优问题演示；全局最优思路与实现。

3.4 带有决策变量边界的最优化问题

单变量最优化问题；多变量最优化问题；基于问题的描述与求解；边界问题全局最优解的尝试。

3.5 最优化问题应用举例

线性回归问题的求解；曲线的最小二乘拟合；边值微分方程的打靶求解；方程求解问题转换为最优化问题。

最优化技术是当前科学研究中一类重要的手段。所谓最优化就是找出使得目标函数值达到最小或最大的自变量值的方法。毫不夸张地说，学会了最优化问题的思想与求解方法，可以将科研的水平提高一个档次，因为原来解决问题得到一个解就满足了，学会了最优化的思想后将很自然地将追求问题最好的解。最优化问题可分为无约束最优化问题和有约束最优化问题。

本章侧重于介绍无约束最优化问题以及 MATLAB 求解方法，在 3.1 节中先给出无约束最优化问题的定义与标准数学模型，然后介绍无约束最优化问题的解析解方法与图解法，并给出全局最优解与局部最优解的定义与判定方法，最后以简单

的一元函数最优化问题为例，介绍最优化问题的算法与MATLAB实现。3.2节侧重于基于MATLAB最优化工具箱函数的最优化问题求解方法，并通过例子演示相关求解函数的使用格式与应用技巧，演示梯度信息在最优化问题求解中的应用与效果，给出基于并行计算的最优化问题求解方法。3.3节探讨一般最优化问题的全局最优解方法，给出一个尝试求解问题全局最优解的思路和其MATLAB实现，并通过一个改进的测试函数检验该算法，验证该算法的有效性。3.4节介绍带有决策变量边界限制的最优化问题及其求解方法，并试图得到单变量与多变量最优化问题的全局最优解。3.5节探讨如何使用最优化技术求解一些实际应用问题。

3.1 无约束最优化问题简介

无约束最优化（unconstrained optimization）问题是最常见也是最简单的一类最优化问题。本节首先介绍无约束最优化问题的标准数学模型，并探讨最优化问题的解析解方法与图解方法，给出全局最优解与局部最优解的概念，以及简单一元最优化问题的求解算法并通过例子演示其MATLAB实现。

3.1.1 无约束最优化问题的数学模型

定义 3-1 ► 无约束最优化

无约束最优化问题的一般数学描述为

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{3-1-1}$$

其中，$\boldsymbol{x}=[x_1,x_2,\cdots,x_n]^{\mathrm{T}}$ 称为决策变量；标量函数 $f(\cdot)$ 称为目标函数。

上述数学描述的物理含义是如何找出求取一组 $\boldsymbol{x}$ 向量，使得目标函数 $f(\boldsymbol{x})$ 的值为最小，故该问题又称为最小化问题。由于这里的 $\boldsymbol{x}$ 向量的值可以任取，所以这类最优化问题又称为无约束最优化问题。

其实，这里给出的最小化是最优化问题的通用描述，不失普遍性。若要想求解最大化问题，那么只需给目标函数 $f(\boldsymbol{x})$ 乘以 -1 就能立即将其转换成最小化问题，所以本章及后续介绍中描述的全部问题都只考虑最小化问题，非最小化问题需要事先转换成最小化标准型。

3.1.2 无约束最优化问题的解析解方法

无约束最优化问题的最优点 $\boldsymbol{x}^*$ 处，目标函数 $f(\boldsymbol{x})$ 对 $\boldsymbol{x}$ 各个分量的一阶导数为0，从而可以列出下面的方程。

$$\left.\frac{\partial f}{\partial x_1}\right|_{\boldsymbol{x}=\boldsymbol{x}^*}=0,\quad \left.\frac{\partial f}{\partial x_2}\right|_{\boldsymbol{x}=\boldsymbol{x}^*}=0,\quad \cdots,\quad \left.\frac{\partial f}{\partial x_n}\right|_{\boldsymbol{x}=\boldsymbol{x}^*}=0 \tag{3-1-2}$$

求解这些方程构成的联立方程可以得出极值点。其实，解出的一阶导数均为 0 的极值点不一定都是极小值的点，其中有的还可能是极大值点。极小值问题还应该有正的二阶导数。对于单变量的最优化问题，可以考虑采用解析解方法进行求解。然而用解析解方法求解多变量最优化问题，因为需要转换成求解多元非线性方程，其难度甚至高于直接最优化问题，所以没有必要采用该方法。

3.1.3　无约束最优化问题的图解法

如果一元方程或二元方程组已知，则可以将方程用匿名函数或符号表达式描述出来，然后采用 `fplot()` 或 `fimplicit()` 函数将方程曲线绘制出来，再通过读取曲线的交点信息即可以获得方程的解。

例 3-1　考虑一元函数 $f(t)=\mathrm{e}^{-3t}\sin(4t+2)+4\mathrm{e}^{-0.5t}\cos(2t)-0.5$，试用解析解方法和图形求解的方法研究该函数的最优性。

解　可以先表示该函数，并用解析解方法求解该函数的一阶导数，用 `fplot()` 函数可以绘制出 $t\in[0,4]$ 区间内原函数与一阶导函数的曲线，如图 3-1 所示。

```
>> syms t;
   y=exp(-3*t)*sin(4*t+2)+4*exp(-0.5*t)*cos(2*t)-0.5; %描述目标函数
   y1=diff(y,t)                                       %求取目标函数的一阶导函数
   fplot([y,y1],[0,4]), line([0,4],[0,0])             %绘制一阶导函数曲线
```

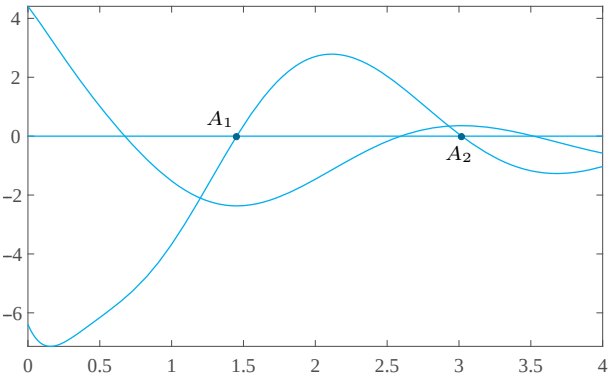

图 3-1　一元函数的导数和方程图解法

通过下面的计算可以得出导数函数为

$$f'(t)=-\mathrm{e}^{-3t}(3\sin(4t+2)-4\cos(4t+2))-2\mathrm{e}^{-t/2}(\cos 2t+4\sin 2t)$$

求解方程 $f'(t)=0$，则可以得出方程的两个解为 $x_1=1.4528, x_2=3.0190$。对原函数求二阶导数，则这两个点的导函数值分别为 $z_1=7.8553, z_2=-3.646$。

```
>> x1=vpasolve(y1,2), x2=vpasolve(y1,3)
   y2=diff(y,2); z1=subs(y2,t,x1), z2=subs(y2,t,x2)
```

其实，求解导函数等于0的方程不比直接求解其最优值简单。用图解法可以看出，在这个区间内有两个点，即 A_1 和 A_2，使得它们的一阶导函数为0，但从其一阶导数走向看，A_2 点对应负的二阶导数值，所以该点对应于极大值点，而 A_1 点对应于正的二阶导数值，故为极小值点。

然而因为给定的函数是非线性函数，所以用解析解方法或类似的方法求解最小值问题一点都不比直接求解最优化问题简单。因此，除演示之外，不建议用这样的方法求解该问题，建议直接采用最优化问题求解程序得出问题的解。

3.1.4 局部最优解与全局最优解

一般的数值最优解方法都采用搜索方法，用户可以给出一个初始搜索点，从这个搜索点出发，采用不同的数值解算法，根据目标函数的实际情况找到下一步的一个点，再根据该点决策变量的目标函数值搜索下一个点。很显然，可以考虑迭代的方法一步一步地搜索出最优的点，使得目标函数的值最小。

对一元问题而言，由于目标函数可以表示为曲线形式，所以一般搜索方法可以形象地理解成在初始搜索点处放置一个小球，让小球沿曲线滚下，这样最终小球将在某一个点处停下来，这时，小球的速度为0，即在这点处目标函数的导数为0，这样的点就是期望的最优点。下面将给出一个例子，介绍数值最优化中局部最优值与全局最优值的概念。

例 3-2 假设目标函数为 $y(t)=\mathrm{e}^{-2t}\cos 10t+\mathrm{e}^{-3t-6}\sin 2t, 0\leqslant t\leqslant 2.5$，试观察不同的初值能得出的最小值，并讨论局部最小值与全局最小值的概念。

解 可以由下面的语句直接绘制出目标函数在感兴趣区域的曲线，如图3-2所示。

```
>> f=@(t)exp(-2*t).*cos(10*t)+exp(-3*(t+2)).*sin(2*t);    % 目标函数
   fplot(f,[0,2.5]);
```

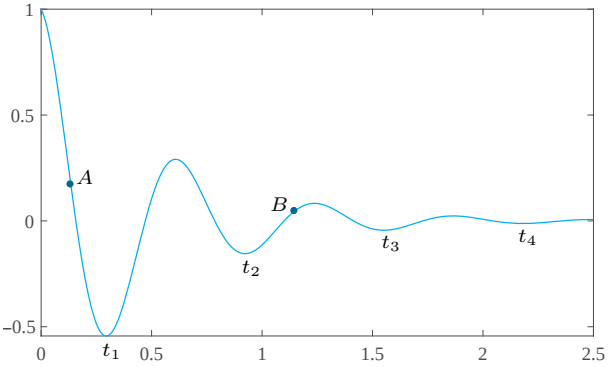

图3-2 函数定义域为 $t\geqslant 0$ 的曲线

如果在 A 点放置一个小球(即将 A 点设为初始搜索点)，则小球滚下后经过反复滚动，最终可能停止在 t_1 点。如果初始点数值为 B 点，则小球经过滚动后最终收敛到 t_2

点，该点目标函数的一阶导数也是 0，所以该点也是原问题的最优解。如果初始值选择在 t 较大的位置，可能找到的最优值还可能是 t_3 或 t_4。

从例 3-2 可以看出，一个已知的目标函数可能有多个最优值，但其目标函数的值可能是不同的。在这个例子中，从给定的区间看，t_1 点称为原问题的全局最优解，因为该点对应的目标函数值是最小的。其他各点，t_2、t_3 和 t_4 又称为问题的局部最优解。

定义 3-2 ▶ 局部最优解

如果在一个邻域 C 内存在 $\boldsymbol{x}^*$ 点，使得 $f(\boldsymbol{x}^*) \leqslant f(\boldsymbol{x})$ 对所有 $\boldsymbol{x} \in C$ 成立，则 $\boldsymbol{x}^*$ 称为局部最优解。

定义 3-3 ▶ 全局最优解

如果在实数域内存在一个 $\boldsymbol{x}^*$ 点，使得在定义域内 $f(\boldsymbol{x}^*) \leqslant f(\boldsymbol{x})$ 成立，则决策变量 $\boldsymbol{x}^*$ 称为无约束最优化问题的全局最优解。

从实际问题求解的角度看，人们追求的是问题的全局最优解，而不是局部最优解，如何获得问题的全局最优解是研究者普遍感兴趣的问题。事实上，目前所有的最优化算法没有哪一种能保证能求出最优化问题的全局最优解，只能说“更可能”获得全局最优解。在实际应用中，局部最优解没有什么价值，因为其他很多非最优解处的函数值都可能优于局部最优解处的目标函数值。例如，在图 3-2 曲线上看，$t \in (0.174, 0.434)$ 处的函数值都优于局部最优解 t_2。

例 3-3　重新考虑例 3-2 中的问题，如果感兴趣的区域变成了 $(-0.5, 2.5)$，试重新分析问题的全局最优解与局部最优解。

解　现在再考虑更大些的定义域，即 $t \geqslant -0.5$，则用下面的语句能绘制出该函数在新定义域内的曲线，如图 3-3 所示。由得出的曲线看，可能的最优解为 t_0，t_1，t_2，t_3 和 t_4。这时的 t_1 点已经不再是问题的全局最优值，而是一个局部最优值，新的全局最优值变成了 t_0。从这个例子可以看出，同样的目标函数随着感兴趣区域的不同，可能有不同的全局最优解。

```
>> f=@(t)exp(-2*t).*cos(10*t)+exp(-3*(t+2)).*sin(2*t);
   fplot(f,[-0.5,2.5]); ylim([-2,1.2])
```

若将定义域扩展到 $t \in (-\infty, \infty)$ 区间，则原问题没有真正意义的全局最优解。

3.1.5　数值求解算法的 MATLAB 实现

求解最优化问题有许多种数值算法，本节给出一种简单的方法演示无约束最优化问题的数值求解，并给出 MATLAB 的实现。

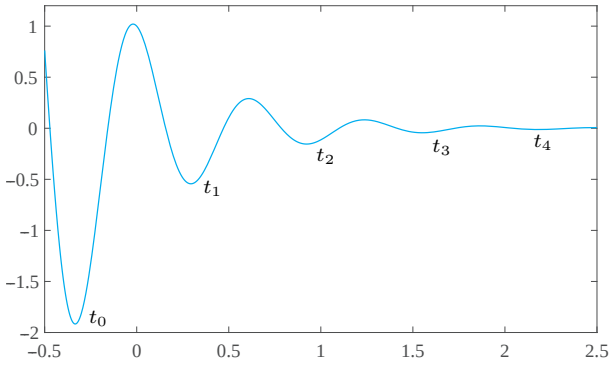

图 3-3　函数定义域为 $t \geqslant -0.5$ 的曲线

通常用到的数值最优化方法都是先选定初值 x_0，然后用迭代方法得出原始问题的数值解。假设已知第 k 步迭代的决策变量值为 x_k，依据 Taylor 级数展开技术，可以用一个二次函数 $g(x)$ 逼近 x 处的函数值[9]。

$$g(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2 \tag{3-1-3}$$

且已知 $f(x_k) = g(x_k), f'(x_k) = g'(x_k), f''(x_k) = g''(x_k)$。现在不优化 $f(x)$ 函数，而考虑简单二次函数 $g(x)$ 的优化问题，该函数最优解存在的条件是 $g'(x) = 0$，由其推导出

$$0 = g'(x) = f'(x_k) + f''(x_k)(x - x_k) \tag{3-1-4}$$

令 $x = x_{k+1}$，则可以推导出

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \tag{3-1-5}$$

如果用后向差分数值导数计算函数的二阶导数，则可以得出迭代公式

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})} f'(x_k) \tag{3-1-6}$$

其实，仔细观察这里的求解方法，不难看出该方法事实上就是求解 $f'(x) = 0$ 代数方程的方法，递推公式类似 Newton–Raphson 迭代法，因为函数的二阶导数采用了后向差分算法取代，从而得出该方程的数值解。下面将通过例子演示如何用 MATLAB 求解一元最优化问题最优值的方法与过程。

例 3-4　试利用这里给出的算法求解例 3-1 中的问题。

解　若想求解这样问题的最优解，则需要用匿名函数描述目标函数的导数，而例 3-1 已经给出了其导函数的解析表达式，所以这里给出简单的匿名函数实现。再选择两个初始参考点 $t_0 = 1, t_1 = 0.5$，这样就可以由循环结构实现迭代过程。可以将循环条件设置为 $|t_{k+1} - t_k| < \epsilon$，并假设 $\epsilon = 10^{-5}$，亦即两次搜索点之间的距离足够小时停止循环。运行下面的语句，就可以得出函数的最优解。

```
>> df=@(t)-exp(-3*t).*(3*sin(4*t+2)-4*cos(4*t+2)) ...
        -2*exp(-t/2).*(cos(2*t)+4*sin(2*t));
   t0=1; t1=0.5; t=[t0,t1];
   while abs(t1-t0)>1e-5
      t2=t1-(t1-t0)/(df(t1)-df(t0))*df(t1);
      t0=t1; t1=t2; t=[t, t2];
   end
```

得出的中间结果如下，最优解为 $t^* = 1.4528$。可以看出，对这里给出的简单问题而言，仅经过几步迭代，就可以得出原始问题的数值最优解。

$$\boldsymbol{t} = [1, 0.5, 1.7388, 1.4503, 1.4534, 1.4528, 1.4528]$$

有了中间结果，就可以利用下面的语句描述目标函数和导函数曲线，标注出中间的搜索点并同时标注辅助线，如图 3-4 所示。从这些点可以大致了解搜索的中间过程。

```
>> f=@(t)exp(-3*t).*sin(4*t+2)+4*exp(-0.5*t).*cos(2*t)-0.5;
   fplot(f,[0,4]), hold on
   fplot(df,[0,4]), plot(t,df(t),'o'), hold off
```

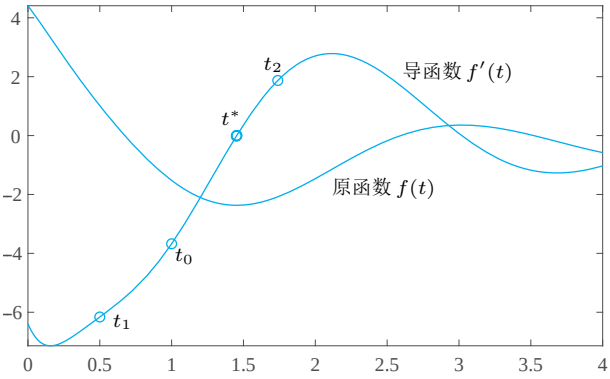

图 3-4　最优值搜索的中间结果

当然，这里仅给出了一元函数最优解的一种常用的搜索方法，对多元问题而言，搜索过程可能要麻烦得多。

例 3-5　试求解 Rosenbrock 函数 $f(x_1, x_2) = 100\left(x_2 - x_1^2\right)^2 + \left(1 - x_1\right)^2$ 的无约束最小值问题。

解　这个问题是英国著名学者 Howard Harry Rosenbrock（1920—2010）构造的用于测试最优化问题求解算法的测试问题。从目标函数可以看出，由于它为两个平方数的和，所以当 $x_2 = x_1 = 1$ 时，整个目标函数有最小值 0。事实上，采用求解导数方程的方式，也可以得出 $x_1 = x_2 = 1$，求出函数的最优值为 0。

```
>> syms x1 x2
```

```
f=100*(x2-x1^2)^2+(1-x1)^2;      %输入目标函数
ff=jacobian(f,[x1,x2]);          %求出目标函数的Jacobi矩阵
[x1,x2]=solve(ff,[x1,x2])        %解一阶导数联立方程,得出最优解
f=100*(x2-x1^2)^2+(1-x1)^2       %计算目标函数的最优值
```

由原始问题可能无法得出函数的一阶导数（对多元问题而言应该为梯度）信息，只知道目标函数本身的信息，求解全局最优化问题可能更加麻烦。本章后续内容将不再介绍这种底层的求解算法，只介绍基于MATLAB最优化工具箱的直接求解方法。

3.2 无约束最优化问题的MATLAB直接求解

MATLAB提供了两个高效的无约束最优化问题求解函数，本节将介绍利用这些函数直接求解无约束最优化问题的方法，并探讨如何提高求解的精度与效率，还将尝试利用并行计算的方法求解最优化问题。

3.2.1 直接求解方法

MATLAB语言中提供了求解无约束最优化问题的函数fminsearch()，其最优化工具箱中还提供了函数fminunc()，二者的调用格式完全一致，为

```
x=fminsearch(Fun,x0)                          %最简求解语句
[x,f0,flag,out]=fminsearch(Fun,x0,opt)        %带有控制选项
[x,f0,flag,out]=fminsearch(Fun,x0,opt,p1,p2,···)  %带有附加参数
```

其中，Fun为描述目标函数的函数句柄，它可以是MATLAB函数的文件名，也可以是匿名函数或inline函数；$\boldsymbol{x}_0$为搜索初值向量；opt为控制选项。除此之外，该函数还允许使用附加参数p_1，p_2，$\cdots$，但不建议使用附加参数，后面将介绍一种替代附加参数的方法。在返回的变量中，$\boldsymbol{x}$为决策变量的最优解；f_0为最优的传递函数值；flag为计算结果的标志，若flag为0，则说明求解过程不成功，flag为1则表示求解成功；返回量out包含一些求解的中间信息，如迭代步数等。

从数值算法看，fminsearch()函数采用了文献[10]中提出的改进单纯形算法，而fminunc()函数可以使用拟Newton算法，也可以使用置信域（trust region）求解算法。从采用的算法看，fminsearch()函数不需要目标函数的梯度信息，拟Newton算法的fminunc()函数不需要目标函数的梯度信息，而fminunc()采用置信域算法时是需要用户提供梯度信息的。

从求解效果上作者发现，对很多算例而言，较新版本的fminunc()函数效率得到了大幅提升，所以建议尽可能使用新版本的求解函数。

下面将通过例子演示无约束最优化问题的数值解法。

例 3-6　已知二元函数 $z=f(x,y)=(x^2-2x)\mathrm{e}^{-x^2-y^2-xy}$，试用 MATLAB 提供的求解函数求出其最小值，并解释解的几何意义。

解　因为函数中给出的自变量是 x 和 y，而最优化函数需要求取的是自变量向量 $\boldsymbol{x}$，故在求解前应该先进行变量替换，如令 $x_1=x, x_2=y$，这样就可以将目标函数手工修改成

$$z=f(\boldsymbol{x})=(x_1^2-2x_1)\mathrm{e}^{-x_1^2-x_2^2-x_1x_2}$$

想求解最优化问题，首先需要将目标函数用 MATLAB 表示出来。通常可以采用两种方法描述目标函数，一种是采用 MATLAB 函数的方法，另一种是匿名函数的方法。先看一下 MATLAB 函数的编程方法，在这种方法下需要由决策向量 $\boldsymbol{x}$ 直接计算目标函数的值 y。

```
function y=c3mopt(x)
   y=(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2));
end
```

将上述函数存储成 c3mopt.m 文件后，就可以由下面的语句直接求解最优化问题，得出最优解为 $\boldsymbol{x}=[0.6111,-0.3056]$，这时返回的标志 flag 为 1，表示求解成功，另外可以看出，在求解过程中调用了 90 次目标函数，总的迭代步数为 46。

```
>> [x,b,flag,c]=fminsearch(@c3mopt,[1;1]) %给出初值并求解最优化问题
```

另一种描述目标函数的方法是采用匿名函数的方法，这样做的一个好处是无须建立一个实体的 MATLAB 函数文件，只需给出动态命令即可；另一个好处是可以直接使用 MATLAB 工作空间中的变量。用下面的语句可以先由匿名函数定义出目标函数，然后求解最优化问题，得出的解与前面的方法完全一致。

```
>> f=@(x)(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2)); %目标函数
   x0=[2; 1]; [x,b,flag,d]=fminsearch(f,x0)  %由初值求解最优化问题
```

同样的问题用 fminunc() 函数求解，可以得出同样的结果。这时，目标函数的调用次数为 66，总的迭代步数为 7。

```
>> [x,b,flag,d]=fminunc(f,[2; 1])  %另一个求解函数
```

比较两种方法，可以看出，用 fminunc() 函数的效率明显高于 fminsearch()，因为对目标函数调用的次数与迭代的步数都明显少于后者，所以在无约束最优化问题求解时，如果安装了最优化工具箱，则建议使用 fminunc() 函数。

其实，利用 surf() 函数可以绘制出目标函数的曲面，如图 3-5 所示。可见，得出的最小值点为得出曲面的谷底。

```
>> syms x y;
   f=(x^2-2*x)*exp(-x^2-y^2-x*y);
   fsurf(f,[-3 3, -2 2])
```

例 3-7　试求解 De Jong 基准测试问题。

$$\min_{\boldsymbol{x}} \sum_{i=1}^{20} x_i^2$$

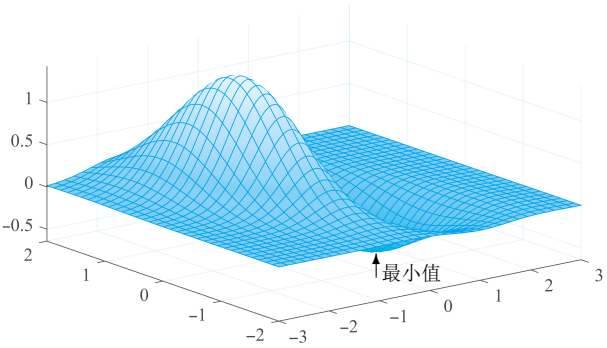

图 3-5　给定函数的三维曲面图

解　很显然，问题的解析解为 $x_i = 0, i = 1, 2, \cdots, 20$。现在可以用匿名函数描述目标函数，然后调用 fminsearch() 函数直接求解最优化问题。其中，在描述目标函数时，$\boldsymbol{x}$(:) 命令将 $\boldsymbol{x}$ 向量转换为列向量，所以这时不论给出的初值 $\boldsymbol{x}_0$ 为行向量还是列向量都能正确地计算目标函数的值。

```
>> f=@(x)x(:)'*x(:);
   x0=ones(20,1); [x0 f0 flag cc]=fminsearch(f,x0)
```

调用上述求解语句，得出目标函数为 $f_0 = 0.0603$，与期望的理论值有较大的差异，且系统给出警告信息“正在退出：超过了函数计算的最大数目，请增大 MaxFunEvals 选项。当前函数值:0.060339”。从给出的提示看，搜索过程非正常结束，原因是目标函数计算的次数大于预设的 MaxFunEvals 选项。同时，应该注意，这时返回的 flag 为 0，说明求解不成功。

3.2.2　最优化控制选项

MATLAB 提供的最优化函数的搜索过程可以这样理解：由用户给定的初值 $\boldsymbol{x}_0$ 进行搜索，每搜索一步，需要计算 $||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k||$ 与 $|f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)|$ 的值，一般情况下，当下面两组条件之一满足，则搜索过程终止。

$$||\boldsymbol{x}_{k+1} - \boldsymbol{x}_k|| < \epsilon_1, \quad |f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k)| < \epsilon_2 \tag{3-2-1}$$

$$k_1 > k_{1\max}, \quad k_2 > k_{2\max} \tag{3-2-2}$$

式中，ϵ_1 和 ϵ_2 为用户指定的误差限，应该选择为很小的正数，这两个参数决定搜索结果的精度；k_1 是目标函数调用次数；$k_{1\max}$ 为用户选定的目标函数计算的最大允许次数；k_2 是实际迭代步数；$k_{2\max}$ 为最大允许的迭代步数，k_1 和 k_2 这两个选项的默认值均为 $200n$，其中，n 为决策变量的个数。

如何修改这些控制选项呢？与前面介绍的解方程问题一样，调用 optimset() 函数，读入控制选项的模板；或使用新版的 optimoptions('fminunc') 命令启动控制选项的设置。这两个函数的控制选项大同小异，但使用的成员变量名不同，可

以调用 optimset() 函数，使用旧版的成员变量名；或调用 optimoptions() 函数，使用旧版或新版的成员变量名。在表 3-1 中给出常用的成员变量名，并给出旧版与新版的对比。

表 3-1　最优化求解函数的成员变量表

旧成员变量名	新成员变量名	成员变量说明
Algorithm	同名	最优化搜索算法的选择与设置，'quasi-newton' 为拟 Newton 算法，而'trust-region' 为置信域算法，前者为默认的算法。注意，置信域算法需要已知目标函数的梯度，否则不能使用
Display	同名	中间结果显示方式，其值可以取'off' 表示不显示中间值，'iter' 表示逐步显示，'notify' 表示在求解不收敛时给出提示（默认选项），'final' 只显示最终值
GradObj	SpecifyObjectiveGradient	求解最优化问题时使用，表示目标函数的梯度是否已知，可以选择为'off' 或'on'，新版使用 true 或 false
MaxIter	MaxIterations	方程求解和优化过程最大允许的迭代步数，若方程未求出解，可以适当增加该值，即 $k_{2\max}$，默认值为 400
MaxFunEvals	MaxFunctionEvaluations	方程函数或目标函数的最大调用次数，即 $k_{1\max}$，默认值为 100 倍决策变量的个数
OutputFcn	同名	常用于中间结果的处理，用户可以编写一个函数，描述在每步迭代中如何处理中间结果，后面将给出演示例子
PlotFcn	PlotFcns	显示寻优收敛过程，建议使用组合 {@optimplotfval, @optimplotfirstorderopt, @optimplotx} 等
TolFun	OptimalityTolerance	误差函数误差限控制量，即 ϵ_2
TolX	StepTolerance	解的误差限控制量，即 ϵ_1
TolCon	ConstraintTolerance	约束条件的误差限
—	UseParallel	是否使用并行求解机制

在一般求解格式的调用语句中，正常情况下，当 ϵ_1 或 ϵ_2 条件满足时搜索过程正常结束，返回的标志 flag 为正数；而 $k_{1\max}$ 与 $k_{2\max}$ 满足时为非正常结束，返回的标志 flag 为 0，所以可以考虑依赖 flag 的值搜索出有意义的最优值。

从给出的成员变量名看，新版的 optimoptions() 函数对应的成员变量名比较复杂，不如原版 optimset() 成员变量名简洁。本书建议尽量采用 optimoptions() 函数设置控制选项，而成员变量名可以考虑使用旧版。注意，调用 optimoptions() 函数时，应该至少给出一个输入变元，建议使用 optimoptions('fminunc') 命令。

下面将通过例子演示成员变量的设置方法与成员变量的效应。

例 3-8　试用不同的算法求解例 3-6 中的无约束最优化问题，并显示中间结果。

解　先考虑采用 fminsearch() 函数直接求解，该函数采用了改进的单纯形法。该求解函数的成员变量设置只能选择 optimset() 函数，不能使用 optimoptions() 函数。

另外，为比较算法，可以设置中间结果的显示，得出的目标函数值为 −0.64142372351。

```
>> f=@(x)(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2));
   ff=optimset; ff.Display='iter';              %成员变量
   x0=[2; 1]; [x,b,c,d]=fminsearch(f,x0,ff)     %单纯形法
```

得出的中间结果如下：

```
 Iteration   Func-count     min f(x)         Procedure
     0           1               0
     1           3               0         initial simplex
     2           5     -0.000641131        expand
     3           7      -0.00181849        expand
     4           9       -0.0132889        expand
     5          11       -0.0654528        expand
     6          13       -0.0835065        reflect
        ... 略去了中间结果
    45          88         -0.641424       contract inside
    46          90         -0.641424       contract inside
```

现在再调用 fminunc() 函数求解最优化问题，其成员变量可以采用 optimset() 函数或 optimoptions() 函数设定，建议使用前者。求解函数采用的默认算法是拟 Newton 算法，调用该函数也可以求解最优化问题，得出的目标函数值为 −0.641423726326，略优于前面的寻优方法。

```
>> ff=optimoptions('fminunc'); ff.Display='iter';
   [x,b,c,d]=fminunc(f,x0,ff)     %默认拟Newton法求解
```

得出的中间结果如下。可以看出，使用该函数迭代步数更少，求解算法更高效。

```
 Iteration  Func-count       f(x)        Step-size      optimality
     0          3               0                        0.00182
     1         24         -0.134531         872.458          0.324
     2         36         -0.134533           0.001          0.324
     3         48         -0.623732             172          0.205
     4         54         -0.641232        0.311866         0.0357
     5         60         -0.641416        0.329315        0.00433
     6         63         -0.641424               1       0.000218
     7         66         -0.641424               1       1.49e-08
```

例 3-9 试求出例 3-7 精确的解。

解 前面介绍过，由于在默认调用格式下系统给出提示，目标函数的计算次数大于最大允许的次数，所以一种很自然的方法是增大 MaxFunEvals，不过这并非是一种好的方法。另一种方法是将得出的 $\boldsymbol{x}_0$ 作为初值重新搜索，再判定得出的 flag 值是否为正，如果为正，则可以终止循环过程，否则继续搜索，直到得出所需的决策变量为止。为得到精确的结果，可以给出更严格的误差限。

```
>> f=@(x)x(:)'*x(:);
   x0=ones(20,1); flag=0; k=0;
   ff=optimset; ff.TolX=eps; ff.TolFun=eps;
   while flag==0, k=k+1     %如果flag为0,则继续搜索
      [x0 f0 flag cc]=fminsearch(f,x0,ff);
   end
   norm(x0), f(x0)
```

可以看出，经过 19 次循环（$k=19$），得到的 $||\boldsymbol{x}_0||=4.0405\times10^{-14}$，目标函数的值低至 $f_0=1.6326\times10^{-27}$，圆满地求解了原始问题。

例 3-10　试利用 fminunc() 函数重新求解例 3-7 中的最优化问题。

解　假设初始搜索点为位于 $(-1000,1000)$ 区间的均匀分布随机数，则可以给出下面的命令，由 fminunc() 函数直接进行寻优计算。

```
>> f=@(x)x(:)'*x(:);
   x0=-1000+2000*rand(20,1);     %随机选择初值
   ff=optimset; ff.TolX=eps; ff.TolFun=eps;
   [x0 f0 flag cc]=fminunc(f,x0,ff), norm(x0)
```

得出 $\boldsymbol{x}_0$ 向量的范数为 $f_0=1.6360\times10^{-11}$，可以看出，得出的结果比较接近理论值。不过，如果使用循环结构，则不会得出更好的求解结果。

从例 3-10 可见，fminunc() 函数可以从给定的初值迅速得出比较精确的结果，然后利用 fminsearch() 函数求精确解，将 $k_{1\max}$ 与 $k_{2\max}$ 设置成大值即可。例如可以尝试下面的语句，这时目标函数的总调用次数为 15877，总耗时为 0.2202 s，误差的范数为 $||\boldsymbol{x}_0||=2.0013\times10^{-12}$。

```
>> x1=-1000+2000*rand(20,1); x0=x1; k=0; tic
   [x0 f0 flag cc]=fminunc(f,x0,ff);
   ff.MaxFunEvals=100000; ff.MaxIter=100000; M=cc.funcCount;
   [x0 f0 flag cc]=fminsearch(f,x0,ff);
   toc, M=M+cc.funcCount, norm(x0)
```

3.2.3　最优搜索中间过程的图形显示

表 3-1 中提及了 OutputFcn 和 PlotFcn 选项，可以用这两个选项显示搜索的动态过程。前者可以设置为响应函数的函数句柄，并编写出响应函数，以便在每一步迭代中显示或处理中间结果；后者可以关联一个预置的显示函数。响应函数的格式与预置函数的显示效果将通过例子演示。

例 3-11　试找出并显示例 3-6 最优化过程的各个中间点。

解　为截取寻优过程的中间点，可以用开关结构编写如下输出处理函数。注意，响应函数的输入、返回变元都应该写成这里给出的固定格式，在寻优过程中 MATLAB 的内在机制会自动生成这些输入变元的值。

```
function stop=c3myout(x,optimValues,state)
stop=false;
switch state                                  %开关结构,监视中间结果
   case 'init', hold on                       %初始化响应:设置坐标系保护
   case 'iter', plot(x(1),x(2),'o'), %迭代响应:将中间结果用圆圈表示
      text(x(1)+0.1,x(2),int2str(optimValues.iteration)); %迭代步数
   case 'done', hold off                      %结束监控过程:取消坐标系保护
end
```

这样就可以在每步迭代中将中间结果标识出来了。要启动这样的监控过程,需要将OutputFcn选项设置为@c3myout。要演示整个优化过程,可以先绘制出原目标函数曲面的等高线图,选择初始搜索点$\boldsymbol{x}_0=[2,1]^{\mathrm{T}}$,用下面的语句开始带有监控的优化过程,这样就可以在等高线上叠印出中间搜索点,如图3-6所示。图中中间搜索点做了编号与标记处理,如果两个中间点的距离特别小,发生重叠(为排版需要已手工移动标记,避免重叠),则说明这时的计算步长很小,接近于收敛值。

```
>> [x,y]=meshgrid(-3:0.1:3, -2:0.1:2);                %生成网格矩阵
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y); contour(x,y,z,30); %等高线图
   f=@(x)(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2));    %目标函数
   ff=optimoptions('fminunc'); ff.OutputFcn=@c3myout;
   x0=[2 1]; x=fminunc(f,x0,ff)                        %解最优化问题
```

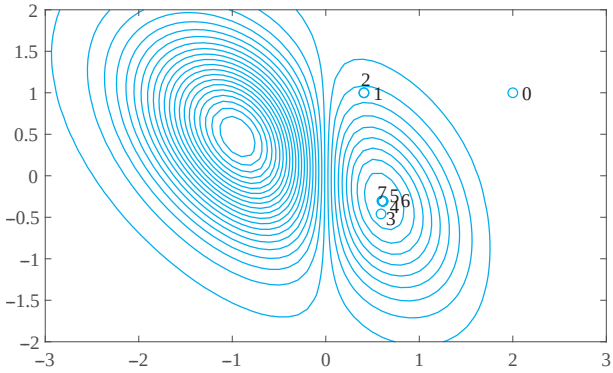

图3-6 求解过程示意图

除了设计OutputFcn响应函数之外,还可以设置PlotFcn成员变量。该成员变量可以关联预置的动态绘图函数。表3-2列出了常用的几个预置显示函数名及其介绍。正常情况下,第二个与第三个预置函数是比较相关的。如果想同时调用几个预置函数,则可以将这些函数名写成单元数组的表示形式。

例3-12 仍考虑例3-6中的最优化过程,试用动态图形显示求解过程。

解 首先考虑一阶最优性的显示。重新求解最优化问题,将得出如图3-7所示的收

表 3-2　寻优过程显示的预置函数

函 数 名	预置函数的解释
@optimplotx	以柱状图的形式动态地运算每个决策变量的演变情况，每步迭代更新一次柱状图的值
@optimplotfval	显示目标函数的变化
@optimplotfirstorderopt	显示一阶最优性的变化情况

敛过程示意图。可见，对这个具体例子而言，收敛过程是很快的。

```
>> ff=optimoptions('fminunc');
   ff.PlotFcn=@optimplotfirstorderopt;
   f=@(x)(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2)); %目标函数
   x0=[2 1]; x=fminunc(f,x0,ff)                          %解最优化问题
```

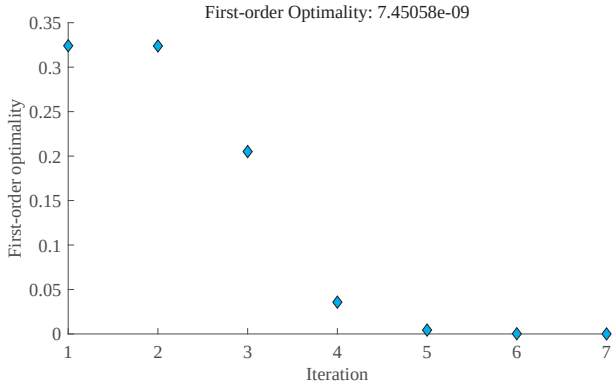

图 3-7　一阶最优性变化过程的示意图

如果调用另外两个预置函数，则可以得出如图 3-8 和图 3-9 所示的动态显示。其中，每步迭代结束后会在图 3-8 中更新当前搜索点的决策变量值。由于本例迭代步数过少，

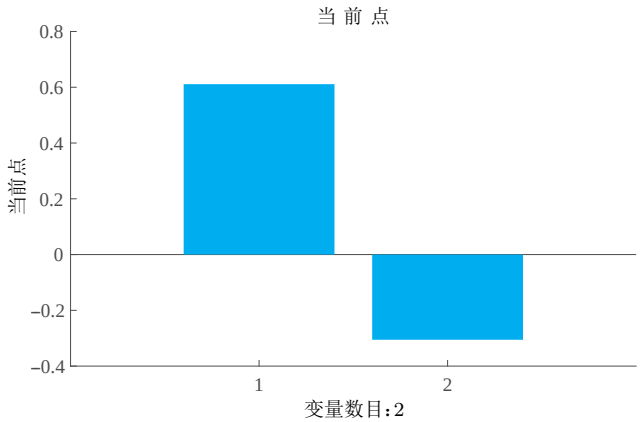

图 3-8　决策变量演变的示意图

这里的动态显示不太明显。图 3-9 中的目标函数变化情况与图 3-7 类似，不过图 3-7 是目标函数一阶导数的变化情况，最终收敛到 0，而图 3-9 是实际目标函数的变化情况，最终收敛到最优的目标函数值。

```
>> ff.PlotFcn={@optimplotx,@optimplotfval};
   x0=[2 1]; x=fminunc(f,x0,ff)                    %解最优化问题
```

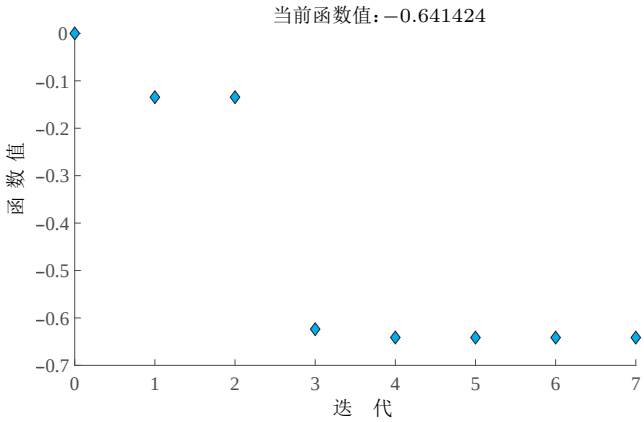

图 3-9 目标函数变化过程示意图

3.2.4 附加参数的传递

在实际的最优化问题中，目标函数除了与 $\boldsymbol{x}$ 有直接关系外，还可以带有其他附加参数，一般情况下，这些附加参数是必要的。本节将演示附加参数的使用方法，还将探讨避免使用附加参数的方法。

例 3-13 已知扩展的 Dixon 问题：

$$\min_{\boldsymbol{x}} \sum_{i=1}^{n/10}\left[(1-x_{10i-9})^2+(1-x_{10i})^2+\sum_{j=10i-9}^{10i-1}(x_j^2-x_{j+1})^2\right]$$

如果 $n=50$，试求解该最优化问题。

解 由给出的式子可见，目标函数的值除了和决策变量 x_i 有关之外，还和 n 有关。因为 n 不能写成决策变量的形式，只能用附加参数的形式写入目标函数。显然，原问题的解析解为 $x_i=1, i=1,2,\cdots,n$。如果 n 是 10 的倍数，则可以将原始的 $\boldsymbol{x}$ 向量转换成 10 行、$n/10$ 列的 $\boldsymbol{X}$ 矩阵，这样，$\boldsymbol{X}$ 矩阵的第 i 列的元素就是决策变量 x_{10i-9} 至 x_{10i} 项的元素值。利用向量化方法就可以编写出下面的 MATLAB 函数描述目标函数。

```
function y=c3mdixon(x,n)                    %注意函数入口
   m=n/10; mustBePositiveInteger(m);        %确保 m 为正整数，否则报错
   X=reshape(x,10,m); y=0;                  %将向量转成矩阵
   for i=1:m
```

```
    y=y+(1-X(1,i))^2+(1-X(10,i))^2+...  %计算(1-x_{10i-9})^2+(1-x_{10i})^2
       sum((X(1:8,i).^2-X(2:9,i)).^2);  %计算Σ(x_j^2-x_{j+1})^2
end, end
```

注意，由于标准目标函数传递的格式下，n 的值并不能直接传入目标函数，所以应该将 n 用作附加参数，这样，该函数的输入变元除了 $\boldsymbol{x}$ 之外，还有第二输入变元 n。

描述了目标函数，就可以考虑由下面的命令直接求解最优化问题。注意，在求解语句调用时，应该给出附加参数 n 的值，与描述目标函数的 m 函数必须保持一致，否则不能直接求解。求解过程中还可以动态地给出决策变量的演变情况，如图 3-10 所示。

```
>> ff=optimoptions('fminunc');
   ff.PlotFcn=@optimplotx;                %动态显示决策变量的演变
   ff.TolX=eps; ff.TolFun=eps;            %给出苛刻的收敛条件
   n=50; x0=rand(n,1);                    %设置随机初值
   [x0,f0,flag,cc]=fminunc(@c3mdixon,x0,ff,n); f0
```

不过该函数可能给出如下警告信息“fminunc stopped because it exceeded the function evaluation limit, options.MaxFunctionEvaluations = 5.000000e+03”，且得出的目标函数为 7.3568×10^{-5}，该值过大，且 flag 值为 0，说明实际目标函数调用次数超过最大允许次数，求解不成功。从图 3-10 中给出的决策变量动态演变图可见，尽管决策变量一致在收敛，但个别决策变量（如 x_{39}）仍然离全局最优值比较远，甚至其值仅有 0.6，而不是理论值 1。

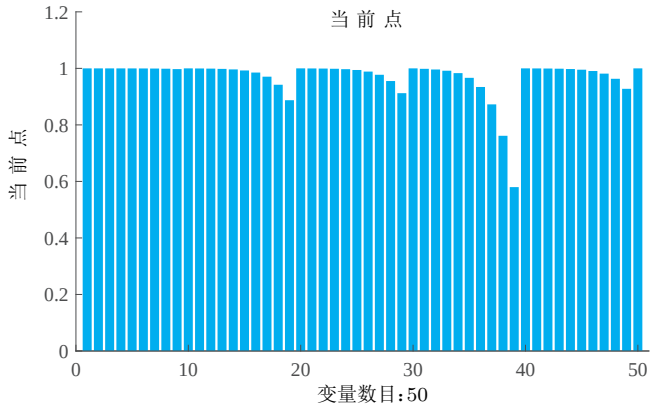

图 3-10　决策变量演变的示意图

可以考虑将求解语句嵌入循环结构，这样经过 4 步循环，可以得到较高精度的数值解，目标函数的值为 8.6509×10^{-8}，得出的 x_i 与理论值的最大误差为 0.0537。尽管本例设置了严苛的误差限，得到的结果误差还是过大。若绘制决策变量的演变图，最终仍然能看出误差。

```
>> flag=0; k=0;
   while flag==0, k=k+1
```

```
    [x0,f0,flag,cc]=fminunc(@c3mdixon,x0,ff,n); f0
end, max(abs(x0-1))
```

在较新版本的最优化工具箱中，很多描述似乎不适合使用附加参数，所以这里考虑一种替代的方法，即为MATLAB文件编写一个匿名函数接口，直接传入附加参数的方法，因为匿名函数可以直接使用MATLAB工作空间中的变量。这里将通过例子演示这样的方法。

例3-14 重新考虑例3-13中的问题，试不采用附加参数重新求解最优化问题。

解 例3-13中曾经给出了描述目标函数的MATLAB函数c3mdixon()，该函数使用了附加参数n。在后面将介绍的结构体调用格式下并不允许使用附加参数，这时应该为其建立一个匿名函数接口，将原来的附加参数在MATLAB工作空间内赋值，这样就可以调用下面的语句重新求解上述无约束最优化问题。

```
>> n=50; clear P                          %求出现有的P变量
   P.objective=@(x)c3mdixon(x,n);         %设计接口,避开附加变量
   P.x0=rand(n,1); P.options=ff; P.solver='fminunc';
   [x0,f0,flag,cc]=fminunc(P)
```

3.2.5 最优化问题的结构体描述

MATLAB最优化工具箱还支持用结构体变量描述最优化问题，这样可以使最优化问题的描述更规范。可以建立一个结构体变量problem，该结构体的各个成员变量在表3-3给出。注意，对无约束最优化问题而言，这四个成员变量必须都给出赋值，否则无法求解。注意，为避免麻烦，建议使用problem变量之前，先用clear命令清除该变量。

表3-3 无约束最优化结构体成员变量表

成员变量名	成员变量说明
objective	目标函数句柄，成员变量名也可以是Objective
x0	初始搜索向量
options	控制选项的设置，可以由problem.options=optimset语句设置成默认的选项，也可像前面例子介绍的那样，修改控制选项，然后将修改后的结构体赋给options
solver	应该将其设置为'fminunc'

完成结构体的描述，就可以由下面的语句直接求解最优化问题的结构体变量problem，得出的结果与前面介绍的函数完全一致。

[x,f_m,flag,out]=fminunc(problem)

注意，fminsearch()并不支持这样的调用格式，只能采用前面给出的一般调用方法使用，求解无约束最优化问题。

例 3-15　试用结构体的方式重新描述并求解例 3-6 中的无约束最优化问题。

解　可以用下面命令建立起最优化问题的结构体变量 P，然后调用 fminunc() 函数即可以直接求解原始问题，得出的结果与例 3-6 中的结果完全一致。

```
>> P.solver='fminunc';
   P.options=optimset;                          %用结构体描述整个问题
   P.objective=@(x)(x(1)^2-2*x(1))*exp(-x(1)^2-x(2)^2-x(1)*x(2));
   P.x0=[2; 1]; [x,b,c,d]=fminunc(P)    %直接求解最优化问题
```

例 3-16　试用结构体格式描述例 3-14 给出的最优化问题，并重新求解该问题。

解　由于结构体的格式并不支持附加参数的使用，所以应该仿照例 3-14 介绍的方法，为目标函数 c3mdixon() 设置一个匿名函数的接口，将 n 的值传入匿名函数，这样就可以用结构体的格式描述原始的无约束最优化问题。可以考虑给出下面的语句重新求解最优化问题，得出的结果与前面的方法完全一致。

```
>> P.solver='fminunc';
   n=50; P.options=optimset;
   P.objective=@(x)c3mdixon(x,n);           %建立匿名函数句柄
   P.x0=rand(n,1); [x,b,c,d]=fminunc(P)    %直接求解最优化问题
```

结构体的另一种生成方式是调用 createOptimProblem() 函数，使用该函数定义结构体时，若采用默认控制选项，则不必另行定义 options 属性。由下面的语句可以直接描述最优化问题，并得出相同的解。

```
>> n=50; f=@(x)c3mdixon(x,n);  %建立匿名函数句柄
   P=createOptimProblem('fminunc','Objective',f,'x0',rand(n,1));
   [x,b,c,d]=fminunc(P)          %直接求解最优化问题
```

3.2.6　梯度信息与求解精度

有时最优化问题求解速度较慢，甚至无法搜索到较精确的最优点，尤其是变量较多的最优化问题，所以需要引入目标函数梯度，以加快计算速度，改进搜索精度。然而，有时计算梯度也是需要时间的，也会影响整个运算速度，所以实际求解时应该考虑是否值得引入梯度的概念。

在利用 MATLAB 最优化工具箱求解最优化问题时，也应该和目标函数在同一函数中描述梯度函数，亦即这时 MATLAB 的目标函数应该返回两个变量，第一个变量仍然表示目标函数，第二个变量可以返回梯度函数。同时，还应该将求解控制变量的 GradObj 属性设置成 'on'，或将新的控制选项 SpecifyObjectiveGradient 设置为 1 或 true，这样就可以利用梯度来求解最优化问题。

例 3-17　试考虑例 3-5 中的 Rosenbrock 函数，并利用 MATLAB 的求解函数直接求出无约束最小值问题的解。

解　用下面语句可以绘制出目标函数的三维等高线图，如图 3-11 所示。

```
>> [x1,x2]=meshgrid(0.5:0.01:1.5);          %生成网格数据
   z=100*(x2.^2-x1).^2+(1-x1).^2;           %计算目标函数
   contour3(x1,x2,z,100), zlim([0,310])     %绘制三维等高线图
```

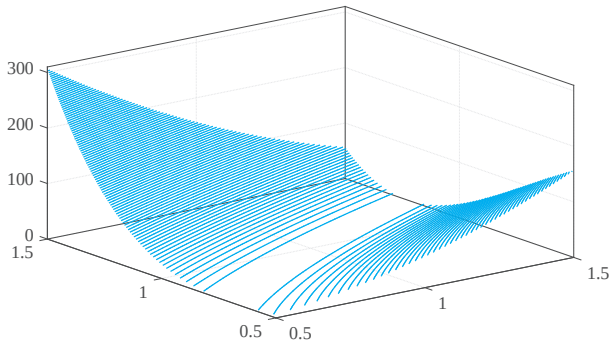

图 3-11　Rosenbrock 目标函数的三维等高线图

由得出的曲线看，其最小值点在图中的一个很窄的白色带状区域内，故 Rosenbrock 目标函数又称为香蕉函数，而在这个区域内的函数值变化较平缓，这就给最优化求值带来很多麻烦。该函数经常用来测试最优化算法的优劣。现在观察用下面的语句求解最优化问题，可以尝试用 fminunc() 函数求解最优化问题，再将结果作为初值继续搜索，经过 100 次循环，即可求解最优化问题。

```
>> f=@(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2; %目标函数的匿名函数描述
   ff=optimoptions('fminunc');
   ff.TolX=eps; ff.TolFun=eps; x=[0;0];
   for k=1:100, k
      [x,f0,flag,cc]=fminunc(f,x,ff)      %最优化问题的数值求解
   end
```

这时得出的最优解为 $\boldsymbol{x}=[0.999995588079578, 0.999991166415393]^{\mathrm{T}}$。可见，即使经过了长时间的运算，该算法也无法精确搜索到真值 $(1,1)$，用传统的最速下降法更无法搜索到真值，所以这时需要引入梯度的概念。对给定的 Rosenbrock 函数，利用符号运算工具箱即可以求出其梯度向量。

```
>> syms x1 x2; f0=100*(x2-x1^2)^2+(1-x1)^2;
   J=jacobian(f0,[x1,x2])      %梯度计算
```

可以求出梯度向量为

$$\boldsymbol{J}=\left[-400(x_2-x_1^2)x_1-2+2x_1, 200x_2-200x_1^2\right]$$

这时，可以在目标函数中描述其梯度，故需要重新编写目标函数为

```
function [y,Gy]=c3fun3(x)
   y=100*(x(2)-x(1)^2)^2+(1-x(1))^2; %需返回两个变量，不能用匿名函数
   Gy=[-400*(x(2)-x(1)^2)*x(1)-2+2*x(1);
```

```
        200*x(2)-200*x(1)^2];
end
```

这样，就应该给出如下命令得出 $\boldsymbol{x}=[1.000000000000012, 1.000000000000023]^{\mathrm{T}}$。

```
>> ff.GradObj='on';                    %启用梯度函数选项
   x=fminunc(@c3fun3,[0;0],ff)         %利用梯度重新求解
```

可见，引入梯度可以明显加快搜索的进度，且最优解也基本逼近真值，这是不使用梯度不可能得到的，所以从本例可以看出梯度在搜索中的作用。然而，在有些例子中引入梯度也不是很必要，因为梯度本身的计算和编程需要更多的时间。

如果用结构体的方式描述最优化问题，则可以得出与前面一致的解。

```
>> problem.solver='fminunc';          %求解器设定
   problem.x0=[2; 1];                 %初值设置
   problem.objective=@c3fun3;         %目标函数设置
   ff=optimset; ff.GradObj='on';      %控制选项设置
   ff.TolX=eps; ff.TolFun=eps; problem.options=ff;
   [x,b,c,d]=fminunc(problem)         %用结构体描述整个问题，直接求解
```

如果不使用梯度信息，但采用 fminsearch() 函数进行求解，则可以得出原问题精确的解。可见，不基于梯度信息的求解方法也可能较好地求解这类问题。

```
>> ff.GradObj='off';
   x=fminsearch(f,[0;0],ff)           %直接求解
```

例 3-18　例 3-13 演示了一个比较复杂的最优化问题，不论选择如何苛刻的控制选项，都无法精确求解该问题，实际误差可能高达 0.001 级。这是因为仅由目标函数难以获得精确的解。试考虑引入梯度信息，重新求解最优化问题。

解　由于原来的决策变量可以每 10 个分成一组，所以可以由下面的命令推导目标函数对每个决策变量的一阶导数。

```
>> syms x [1,10]
   f=(1-x1)^2+(1-x10)^2+sum((x(1:8).^2-x(2:9)).^2);
   df=simplify(jacobian(f,x))     %求解目标函数的梯度
```

可以得出

$$\boldsymbol{d}_f=[2x_1-4x_1(-x_1^2+x_2)-2, d_2, d_3, \cdots, d_8, -2x_8^2+2x_9, 2x_{10}-2]$$

其中，$d_j=-2x_{j-1}^2+2x_j-4x_j(x_{j+1}-x_j^2)$，$j=2,3,\cdots,8$。这里的 x_j 事实上就是 $\boldsymbol{X}(j,i)$。由上面公式不难写出新的目标函数。

```
function [y,J]=c3mdixona(x,n)
   m=n/10; X=reshape(x,10,m); y=0; J=[];
   for i=1:m, j=3:7;
      y=y+(1-X(1,i))^2+(1-X(10,i))^2+...
         sum((X(1:8,i).^2-X(2:9,i)).^2);
      dj=-2*X(j-1,i).^2+2*X(j,i)-...
```

```
            4*X(j,i).*(X(j+1,i)-X(j,i).^2);
        J=[2*X(1,i)-4*X(1,i)*(-X(1,i)^2+X(2,i)-2;
           dj; -2*X(8,i)^2+2*X(9,i); 2*X(10,i)-2];
    end, end
```

在该目标函数中，除了返回目标函数 y 的值之外，还返回目标函数对各个决策变量的一阶导数 $\boldsymbol{J}$，以列向量形式返回。这样，就可以由下面语句直接求解最优化问题，得出的收敛示意图如图 3-12 所示。这时的目标函数达到 4.8071×10^{-31}，决策变量与理论值的最大偏差低至 3.7748×10^{-15}。由此可见，引入梯度信息，可以大幅提升最优化问题的求解精度与求解速度。

```
>> ff=optimoptions('fminunc'); ff.PlotFcn=@optimplotfirstorderopt;
   ff.GradObj='on'; ff.TolX=eps; ff.TolFun=eps;   %设置控制选项
   n=50; x0=rand(n,1);                            %设置随机初值
   [x0,f0,k,cc]=fminunc(@c3mdixona,x0,ff,n); f0   %求解最优化问题
   max(abs(x0-1))                                 %决策变量最大误差
```

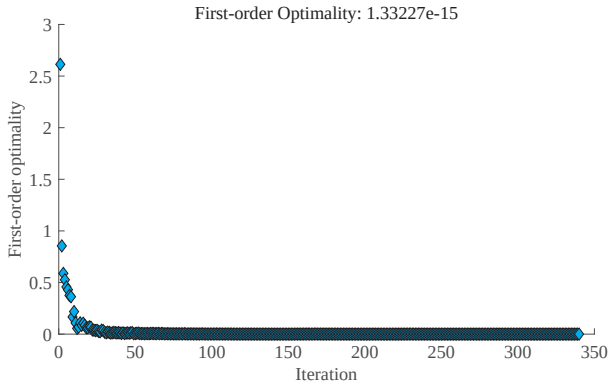

图 3-12　Dixon 目标函数的收敛示意图

需要指出的是，Rosenbrock 函数和 Dixon 函数是为检测寻优算法优劣而建立起来的人造函数，解决该问题的有效方法需要引入目标函数的梯度。实际应用中，很多寻优算法都是无须梯度信息的，用户可以根据需要决定是否引入梯度信息。

例 3-19　重新考虑例 3-7 中介绍的最优化问题，由于采用的最优化算法未利用梯度信息，所以得出的结果还是比较慢的，试利用梯度信息重新求解该问题。

解　由给定的目标函数，可以立即看出目标函数的梯度为 $2[x_1,x_2,\cdots,x_{20}]=2\boldsymbol{x}$，所以可以利用 MATLAB 函数的形式将目标函数与梯度信息表示出来。注意，由于该函数需要返回两个变元，所以不能采用匿名函数的形式描述目标函数与梯度。

```
function [y,G]=c3mdej1(x)
   y=x(:).'*x(:); G=2*x;
end
```

有了相应的信息，则可以由下面的语句直接最优化问题。可以看出，利用梯度信息，则用 3 步迭代即可以得出问题的最优解，$\boldsymbol{x}$ 向量的范数低至 6.6243×10^{-30}，目标函数的值低至 $f_0=4.3881\times10^{-59}$，精度远高于其他数值算法。

```
>> clear problem;
   problem.solver='fminunc';                    %设定求解器
   problem.objective=@c3mdej1;                  %设置目标函数
   ff=optimoptions('fminunc'); ff.TolX=eps; ff.TolFun=eps;
   ff.GradObj='on'; problem.options=ff; %设置控制选项
   problem.x0=-100+200*rand(20,1);              %设置初值
   [x,f0]=fminunc(problem), norm(x)             %用结构体描述问题并求解
```

例 3-20　试利用置信域算法重新求解例 3-6 的最优化问题。

解　如果采用置信域算法，则需要首先获得目标函数的梯度。可以使用符号运算的方式，直接推导目标函数的梯度表达式。

```
>> syms x1 x2;
   f=exp(-x1^2-x1*x2-x2^2)*(x1^2-2*x1)   %输入目标函数的解析表达式
   G=simplify(jacobian(f,[x1,x2]))       %求解 Jacobi 矩阵
```

这样可以直接推导出目标函数的梯度为

$$\boldsymbol{G}(\boldsymbol{x})=\mathrm{e}^{-x_1^2-x_1x_2-x_2^2}\begin{bmatrix}2x_1+2x_1x_2-x_1^2x_2+4x_1^2-2x_1^3-2\\(-x_1^2+2x_1)(x_1+2x_2)\end{bmatrix}$$

有了目标函数与目标函数的梯度数学表达式，就可以建立下面的 MATLAB 函数，直接描述这两个量，通过下面的函数返回。

```
function [f,Gy]=c3mgrad(x)
   u=exp(-x(1)^2-x(1)*x(2)-x(2)^2); f=u*(x(1)^2-2*x(1));
   Gy=[2*x(1)+2*x(1)*x(2)-x(1)^2*x(2)+4*x(1)^2-2*x(1)^3-2;
       (-x(1)^2+2*x(1))*(x(1)+2*x(2))]*u;
end
```

由相关信息构造出最优化问题的结构体变量，可以将求解算法直接设置为置信域算法，然后给出 fminunc() 函数直接求解问题。

```
>> clear problem;
   problem.solver='fminunc';                    %设定求解器
   problem.objective=@c3mgrad;                  %目标函数
   ff=optimoptions('fminunc');
   ff.Algorithm='trust-region'; ff.GradObj='on';
   ff.Display='iter'; problem.options=ff;   %设置控制选项
   problem.x0=[2;1];                            %设置初值
   [x,f0]=fminunc(problem)                      %求解
```

下面是得出的中间结果：

Iteration	f(x)	Norm of step	First-order optimality	CG-iterations
0	0		0.00182	
1	0	10	0.00182	1
2	0	2.5	0.00182	0
3	-0.0120683	0.625	0.0558	0
4	-0.46831	1.25	0.434	1
5	-0.46831	1.25226	0.434	1
6	-0.602878	0.313066	0.252	0
7	-0.640866	0.296542	0.0372	1
8	-0.641424	0.0310125	6.49e-05	1
9	-0.641424	6.10566e-05	6.96e-10	1

从这里给出的例子来看，执行效率显然低于例 3-8 测试的拟 Newton 算法，所以可以得出结论：并不是所有无约束最优化问题都适合使用梯度信息。

3.2.7 基于问题的描述方法

从MATLAB 2017b版的MATLAB开始，最优化工具箱提出了最优化问题的一种新型描述方法：基于问题（problem based）的描述方法。早期的基于问题的描述方法只适合于描述线性或二次型函数，新的版本逐渐适合于更广泛的最优化问题的描述与求解。

对无约束最优化问题而言，需要按照如下步骤描述并求解：

（1）**最优化问题的创建**。可以由 `optimproblem()` 函数创建一个新的空白最优化问题，该函数的基本调用格式如下：

```
prob=optimproblem('ObjectiveSense','max')
```

如果不给出`'ObjectiveSense'`属性，则求解默认的最小值问题。

（2）**决策变量的定义**。可以由 `optimvar()` 函数实现，该语句的一般格式为

`x=optimvar('x',n,m,k)` 或 `x=optimvar('x',[n,m,k])`

其中，n、m 和 k 为三维数组的维数；如果不给出 k，则可以定义出 $n \times m$ 决策矩阵 $\boldsymbol{x}$；若 m 为 1，则可以定义 $n \times 1$ 决策列向量；若不给出维数，则定义标量。

（3）**目标函数的描述**。可以给 `prob` 变量的 `Objective` 成员变量直接赋值。

（4）**初始条件的设置**。若由 `optimvar()` 函数定义了决策变量 x、y、z，则可以对 $\boldsymbol{x}_0.x$、$\boldsymbol{x}_0.y$ 和 $\boldsymbol{x}_0.z$ 进行设置，得出初值的结构体变量 $\boldsymbol{x}_0$。

（5）**最优化问题的求解**。有了 `prob` 变量之后，就可以调用 `sols=solve(prob,`$\boldsymbol{x}_0$`)` 函数直接求解相关的最优化问题，得出的结果将由结构体 `sols` 返回，该结构体的 $\boldsymbol{x}$ 成员变量即为最优化问题的解。对非线性目标函数而言，必须提供初值 $\boldsymbol{x}_0$ 结

构体，否则不能求解。

也可以由 options=optimset() 函数设置控制选项，或由 optimoptions() 函数设置，再由命令 [sols,f_0,flag]=solve(prob,x_0,'options',options) 得出问题的解。

例 3-21　试用基于问题的描述方程重新描述并求解例 3-6 中的最优化问题。

解　先由 optimproblem() 函数创建一个空白的问题模型 P，再由 optimvar() 函数创建决策变量向量。这样，就可以描述目标函数与搜索初值，调用 solve() 函数直接求解最优化问题，得出的结果与目标函数值同例 3-6 完全一致，得出的 flag 为字符串 'OptimalSolution'，表示求解成功。

```
>> P=optimproblem;                          %创建最优化模型
   x=optimvar('x',2,1);                     %定义决策变量
   P.Objective=(x(1)^2-2*x(1))*...
          exp(-x(1)^2-x(2)^2-x(1)*x(2));    %目标函数描述
   x0.x=rand(2,1);                          %初值设定
   [sol,f0,flag]=solve(P,x0)                %直接求解最优化问题
   sol.x                                    %提取问题的解
```

由 show() 函数还可以由可读的方式显示模型。

```
>> show(P)      %以可读的方式显示最优化问题
```

显示的可读形式为

```
OptimizationProblem:
    Solve for:
       x
    minimize:
       ((x(1).^2 - (2 .* x(1))) .* exp((((-x(1).^2)
     - x(2).^2) - (x(1) .* x(2)))))
```

这里给出的基于问题的描述方法看起来似乎有局限性。如果由 optimvar() 函数定义了 $\boldsymbol{x}$ 变量，则在描述目标函数时不能直接使用 MATLAB 函数或匿名函数。可以调用 fcn2optimexpr() 函数，将 optimvar() 定义的变量与 MATLAB 函数建立关联。下面通过例子演示基于问题的复杂最优化问题的描述与求解，并演示该方法的优势。

例 3-22　试用基于问题的方法重新描述并求解例 3-14 给出的最优化问题。

解　因为目标函数需要调用 c3mdixon.m 文件，所以不能直接使用 P.Objective=c3mdixon($\boldsymbol{x}$,n) 命令定义目标函数。因为 $\boldsymbol{x}$ 不是双精度向量，而是最优化变量，所以不能直接调用 MATLAB 函数，必须通过 fcn2optimexpr() 函数进行转换，具体格式参见后面的例子。此外，成员变量 Objective 不能写成 objective。

```
>> n=50;
```

```
P=optimproblem;                                  %创建最优化模型
x=optimvar('x',n,1);                             %定义决策变量
P.Objective=fcn2optimexpr(@c3mdixon,x,n);        %描述目标函数
x0.x=rand(n,1);                                  %设定初值
[sol,f0,flag]=solve(P,x0)                        %直接求解最优化问题
max(abs(sol.x-1))                                %提取问题的解
```

由这里给出的方法得出的最大误差为 1.0926×10^{-11}，对应的目标函数的值为 1.1935×10^{-22}。可以看出，这里给出的方法得出的结果远远优于 fminunc() 函数。

例 3-23　考虑下面的多维 Rosenbrock 问题，试利用基于问题的方法描述并求解最优化问题。

$$J=\sum_{i=1}^{10}100\left(y_i-x_i^2\right)^2+\left(1-x_i\right)^2$$

解　如果想用传统方法求解这个最优化问题，需要重新选择决策变量，使得决策变量变成单一的向量，再利用手工变换的方法，将目标函数变换成新决策变量的函数，这样的手工转换方法比较麻烦且容易出错。如果采用基于问题的描述方法，则可以构造两个决策变量 $\boldsymbol{x}$、$\boldsymbol{y}$，然后写出目标函数，再求解最优化问题。可以看出，这样描述的目标函数更简洁，求解过程也更直观。

```
>> n=10;
   P=optimproblem;                                  %创建最优化模型
   x=optimvar('x',n,1);                             %定义决策变量
   y=optimvar('y',n,1);                             %定义另一决策变量
   P.Objective=sum(100*(y-x.^2).^2+(1-x).^2);       %描述目标函数
   x0.x=10*rand(n,1); x0.y=10*rand(n,1);            %设定初值
   [sol,f0,flag]=solve(P,x0)                        %直接求解最优化问题
   max(abs(sol.x-1)), max(abs(sol.y-1))             %提取并检验问题的解
```

由上面的语句可见，这时得出的目标函数值为 2.5884×10^{-24}，得出的 $\boldsymbol{x}$ 向量误差为 1.3050×10^{-12}，$\boldsymbol{y}$ 向量的误差为 2.6192×10^{-12}。可以看出，这样的结果远远优于 fminunc() 函数的结果，其精度接近利用梯度信息得出的结果。

3.2.8　离散点最优化问题的求解

在实际应用中，有时某一个需要优化的目标函数原型是未知的，只有一些相应的、离散分布的样本数据点，这时，就可以采用样条插值或其他插值方法拟合目标函数，从而优化目标函数。下面分别介绍一元、二元函数甚至多元函数的插值方法，由这些插值方法即可以计算出感兴趣点的目标函数值。

（1）一元问题。如果已知样本点 $\boldsymbol{x}_0$ 和 $\boldsymbol{y}_0$，则对于任意的 $\boldsymbol{x}$ 向量，可以由匿名函数 f=@($\boldsymbol{x}$)interp1($\boldsymbol{x}_0$,$\boldsymbol{y}_0$,$\boldsymbol{x}$,'spline') 计算出 $\boldsymbol{x}$ 向量各点的函数插值结果。

（2）二元问题。如果已知样本点向量 $\boldsymbol{x}_0$、$\boldsymbol{y}_0$ 和 $\boldsymbol{z}_0$，则可以令 $p_1=x$，$p_2=y$，这

样，函数值的插值结果就可以由函数句柄表示。

f=@($\boldsymbol{p}$)griddata($\boldsymbol{x}_0$,$\boldsymbol{y}_0$,$\boldsymbol{z}_0$,$\boldsymbol{p}$(1),$\boldsymbol{p}$(2),'v4')

（3）**三元问题**。如果已知样本点向量$\boldsymbol{x}_0$、$\boldsymbol{y}_0$、$\boldsymbol{z}_0$与$\boldsymbol{v}_0$，则可以令$p_1 = x$，$p_2 = y$，$p_3 = z$，这样，函数值的插值结果可以由函数句柄表示。

f=@($\boldsymbol{p}$)griddata($\boldsymbol{x}_0$,$\boldsymbol{y}_0$,$\boldsymbol{z}_0$,$\boldsymbol{v}_0$,$\boldsymbol{p}$(1),$\boldsymbol{p}$(2),$\boldsymbol{p}$(3),'v4')

（4）多元函数的散点插值还可以考虑使用griddatan()函数直接构造。

f=@($\boldsymbol{p}$)griddatan([$\boldsymbol{x}_1$(:),$\boldsymbol{x}_2$(:),$\cdots$,$\boldsymbol{x}_m$(:)],$\boldsymbol{z}$,$\boldsymbol{p}$)

有了函数值，就可以调用fminunc()等函数直接求解最优化问题。下面将通过例子演示这样的最优化问题求解方法。

例 3-24　重新考虑例 3-6 中的函数，试由已知函数在$x \in [-3,3]$，$y \in [-2,2]$时生成均匀分布的 200 个样本点，然后仅利用这些离散的散点求出对应函数的最小值，并检验所得出的结果。

解　仿照前面的例子，可以首先生成一些离散点，再由这些离散点通过插值方法构造目标函数的匿名函数，对该匿名函数进行优化，就可以得出最优解为$x = 0.6069$，$y = -0.3085$。可以看出，这样得到的最优点很接近例 3-6 由目标函数表达式得出的最优解，所以这里给出的优化方法是可行的。

```
>> x=-3+6*rand(200,1); y=-2+4*rand(200,1);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);      %生成散点数据
   f=@(p)griddata(x,y,z,p(1),p(2),'v4');    %重建目标函数
   x=fminunc(f,[0,0])                       %由散点数据进行寻优
```

3.2.9　最优化问题的并行求解

通常，大规模最优化问题可能涉及巨大的运算量，所以在实际应用中可以考虑最优化工具箱提供的自动并行计算功能，这需要如下设置控制选项。

```
options=optimoptions('fminunc','UseParallel',true)
```

如果并行计算的选项开启，则求解机制会自动使用并行运算机制，利用并行计算的方法求解最优化问题；如果关闭并行计算机制，则采用普通的方法求解最优化问题。下面将给出较大规模最优化问题的求解实例，不过从无约束最优化问题求解看，开启并行计算机制的意义并不是很大。

例 3-25　考虑例 3-14 中给出的最优化问题，如果$n = 2000$，试比较使用并行计算与不使用并行计算的求解方法。

解　由于并行计算机制的启动本身很耗时，公平起见，如果使用并行计算的方法，先单独启动并行机制，该过程耗时 74.32 s。

```
>> tic, p=parpool(4), toc
```

启动了并行计算机制，就需要给出下面的语句，直接求解最优化问题。事实上，即使不事先启动，最优化函数也会自动启动并行计算机制。启动了并行计算机制，后续计算步骤实际耗时为 18.47 s，得出的误差为 1.0505×10^{-8}。

```
>> problem.solver='fminunc'; n=2000;
   ff=optimoptions('fminunc');
   ff.TolFun=eps; ff.TolX=eps; ff.GradObj='on';
   ff.MaxFunEvals=1000000; ff.MaxIter=1000000;
   ff.UseParallel=true; problem.options=ff; %使用并行计算
   problem.objective=@(x)c3mdixona(x,n);    %建立匿名函数句柄
   x0=rand(n,1); problem.x0=x0;             %选择随机初值
   tic, [x,b,c,d]=fminunc(problem); toc     %求解最优化问题
   max(abs(x-1))                            %决策变量的最大误差
```

如果不采用并行计算，则可以给出下面语句直接求解。为对比方便，采用了相同的初值，这时得出的误差与前面得出的完全一致，耗时为 17.25 s，与使用并行计算的纯计算步骤耗时相仿。不过，采用并行计算机制还应该再加上启动并行计算机制所需的时间，因而耗时远远高于常规的最优化计算。对此例而言并行计算并没有优势。

```
>> delete(p)
   ff.UseParallel=false; problem.options=ff;  %关闭并行计算机制
   tic, [x,b,c,d]=fminunc(problem); toc       %重新计算
   max(abs(x-1))                              %决策变量最大误差
```

3.3 全局最优解的尝试

3.3.1 全局最优问题演示

前面介绍了全局最优解问题。本节将给出一个多谷底的基准测试函数，首先演示该测试函数最优解对初始值的依赖性，然后编写一个能尝试求出全局最优解的 MATLAB 函数，并探讨其求解效率。

例 3-26 考虑一个著名的无约束最优化问题的基准测试函数（benchmark problem）——Rastrigin 函数[11]。

$$f(x_1,x_2)=20+x_1^2+x_2^2-10(\cos 2\pi x_1+\cos 2\pi x_2)$$

试绘制出目标函数的表面图，并用简单的最优化求解函数求解这样的问题，看看会发生什么。

解 目标函数的表面图可以由下面的语句直接得出，如图 3-13 所示。可以看出，表面图凹凸不平，其中有很多波峰与波谷。

```
>> f=@(x1,x2)20+x1.^2+x2.^2-10*(cos(2*pi*x1)+cos(2*pi*x2));
   fsurf(f)      %绘制目标函数的表面图
```

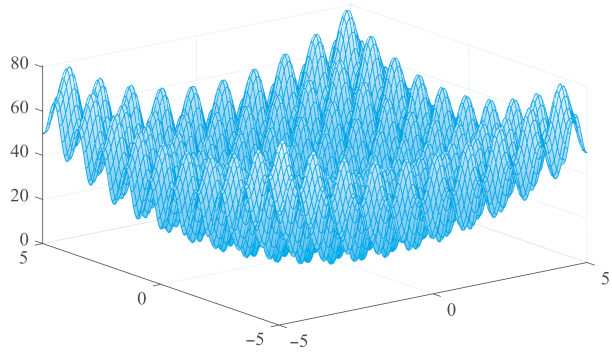

图 3-13　Rastrigin 函数的表面图

如果绘制目标函数的正视图与侧视图(见图 3-14,由于目标函数关于 x_1、x_2 是对称的,所以正视图与侧视图完全一致),可以发现,不同的“最优值”对应不同的目标函数值,$x_1 = x_2 = 0$ 点是原始问题的全局最优解,其他组合都是局部最优解。

```
>> view(0,0)                          %绘制目标函数的正视图
   figure, fsurf(f), view(90,0)       %绘制目标函数的侧视图
```

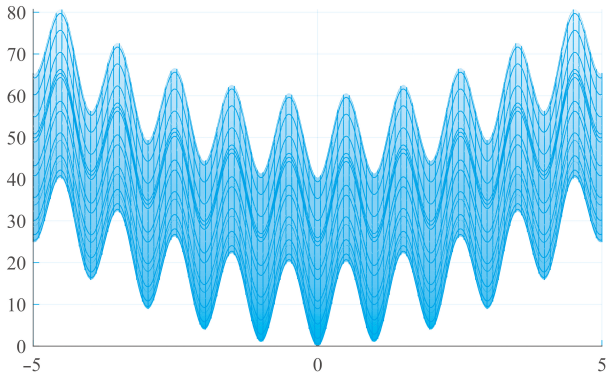

图 3-14　目标函数的正视图、侧视图(二者完全一致)

还可以得出目标函数的等高线图,如图 3-15 所示。从给出的图形可见,最中间的点是全局最小点,另外还有很多波谷点,但它们都是局部最小点。另外,全局最优点附近的几个点可以认为是次最优(subminimum)点。

```
>> fcontour(f,'MeshDensity',1000) %绘制等高线图
```

选择几个不同的初始搜索点,可以由下面的语句得出不同的优化结果。在重新定义目标函数时,没有必要全盘改写原有的匿名函数,只需在原来函数的基础上写一个新的接口函数即可。有了目标函数的匿名函数描述,就可以在以下几个初值下直接寻优计算,得出寻优结果。

```
>> f1=@(x)f(x(1),x(2));  %可以在前面匿名函数的基础上定义新函数
   x1=fminunc(f1,[2,3]), f1(x1), x2=fminunc(f1,[-1,2]), f1(x2)
```

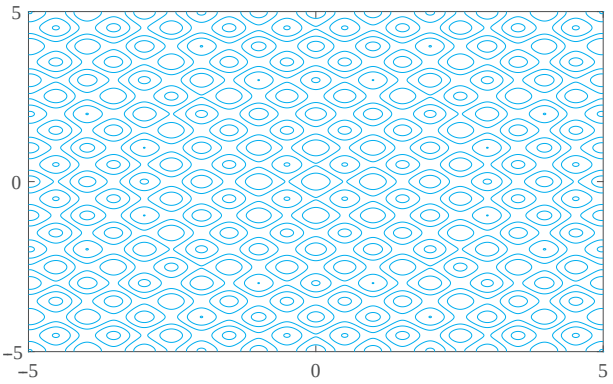

图 3-15　目标函数的等高线图

```
x3=fminunc(f1,[8 2]), f1(x3), x4=fminunc(f1,[-4,6]), f1(x4)
```

得到如下几个结果：

$\boldsymbol{x}_1=[1.9602, 1.9602],\ f(\boldsymbol{x}_1)=7.8409,\ \ \boldsymbol{x}_2=[-0.0000, 1.9602], f(\boldsymbol{x}_2)=3.9205$

$\boldsymbol{x}_3=[7.8338, 1.9602],\ f(\boldsymbol{x}_3)=66.6213,\ \boldsymbol{x}_4=[-3.9197, 5.8779],\ f(\boldsymbol{x}_4)=50.9570$

可以看出，这样得出的最优化结果都是“最优”的，但有显著差异，大多数点为局部最小值点。可以看出，采用传统的最优化搜索方法，如果初始值选择不当，很可能陷入局部最小值。另外，上面并未得到全局最优解 $\boldsymbol{x}=[0,0]$。

3.3.2　全局最优思路与实现

为避免局部最小值问题，经常采用某些智能优化方法，如遗传算法（genetic algorithm）或其他进化类算法，这类方法将在后续章节中给出概略性的介绍。不过即使遗传算法之类的方法也不能确保得到全局最优解，只不过进化类算法号称更可能得出全局最优解。

类似于前面介绍的方程求解的思路，可以采用下面的新算法做全局寻优。首先，用随机的方式在感兴趣区域 (a,b) 选择初值，通过普通的搜索方法得出最优解 $\boldsymbol{x}$，并得出最优目标函数 $f_1=f(\boldsymbol{x})$，如果得出的最优目标值比已经得到的还小，则记录该最优值。重复 N 次这类求解过程，就可能得出问题的全局最优解。基于此思路，可以编写出如下 MATLAB 函数求解全局最优化问题。

```
function [x,f0]=fminunc_global(f,a,b,n,N,varargin)
   arguments, f, a(1,1) double
      b(1,1) double {mustBeGreaterThan(b,a)}
      n(1,1) {mustBePositiveInteger}
      N(1,1) {mustBePositiveInteger}
   end
   arguments (Repeating), varargin; end
```

```
    k0=0; f0=Inf;
    if isstruct(f), k0=1; end     %可以用结构体描述
    for i=1:N
       x0=a+(b-a).*rand(n,1);     %用循环结构生成随机初始搜索点
       if k0==1                   %结构体描述问题求解
          f.x0=x0; [x1 f1 key]=fminunc(f);
       else                       %一般无约束最优化的求解
          [x1 f1 key]=fminunc(f,x0,varargin{:});
       end
       if key>0&&f1<f0, x=x1; f0=f1; end %若得到更好的解则更新
end, end
```

该函数的调用格式为

[$\boldsymbol{x}$,f_0]=fminunc_global(fun,a,b,n,N)

[$\boldsymbol{x}$,f_0]=fminunc_global(fun,a,b,n,N,$\boldsymbol{x}_0$,options)

式中，fun为描述目标函数的MATLAB函数，它可以是匿名函数也可以是MATLAB函数，还可以是描述整个优化问题的结构体变量；a和b为决策变量可能的区间；n为自变量的个数；N为尝试的次数。如果N的选择得当，则返回的变量$\boldsymbol{x}$与f_0很可能是原始最优化问题的全局最优解。如果需要，a和b还可以选择为向量。

需要指出的是，虽然函数调用时给出了a和b参数，这只是自动生成初值的范围，得出的最终解很可能超出这个范围。

例3-27　考虑例3-26中的无约束最优化问题，试求出最优化问题的全局最优解，并评价这里给出的最优解方法。

解　如果将尝试的次数N选择为50，可以看出，每次运行这个函数都能找到全局最优解$x_1=x_2=0$。

```
>> f=@(x)20+x(1)^2+x(2)^2-10*(cos(2*pi*x(1))+cos(2*pi*x(2)));
   [x,f0]=fminunc_global(f,-2*pi,2*pi,2,50) %尝试获得全局最优解
```

为进一步演示这样的全局最优解求解过程，可以循环调用100次这一求解程序，可以看到，每次都能找到全局最优解。

```
>> F=[]; N=50;     %建立一个目标函数值的存储向量，初始值为空白向量
   for i=1:100     %调用求解函数100次并评估找到全局最优解的成功率
      [x,f0]=fminunc_global(f,-2*pi,2*pi,2,N); F=[F,f0];
   end
```

当然，由于使用了均匀分布的随机数，这样的全局最优解$x_1=x_2=0$很容易被找到，所以用这个例子评估全局优化算法并不公平，后面将试图给出更公平的测试函数评价全局最优算法。

例3-28　假设将经典的Rastrigin函数修改成

$$f(x_1,x_2)=20+\left(\frac{x_1}{30}-1\right)^2+\left(\frac{x_2}{20}-1\right)^2-10\left[\cos 2\pi\left(\frac{x_1}{30}-1\right)+\cos 2\pi\left(\frac{x_2}{20}-1\right)\right]$$

可以运行100次这个求解程序，测试一下找到全局最优解的成功率是多少。

解 显然，经过这样的改写，可以得出最优化问题全局最优解的理论值为 $x_1=30$，$x_2=20$。如果将感兴趣搜索区间扩展到 $[-100,100]$，则可以进行如下测试。

```
>> f=@(x)20+(x(1)/30-1)^2+(x(2)/20-1)^2-...
        10*(cos(2*pi*(x(1)/30-1))+cos(2*pi*(x(2)/20-1)));
   F=[]; tic, N=100;
   for i=1:100        %运行该求解函数100次
      [x,f0]=fminunc_global(f,-100,100,2,N); F=[F,f0]; %记录最优值
   end, toc
   sum(F>0.1)         %统计未得出全局最优解的次数
```

可以看出，这次执行100次寻优，有21次没有找到全局最优值 $(30,20)$，其余的79次都找到了全局最优点，成功率79%，总耗时27.77 s。还可以看出，这里给出的函数 `fminunc_global()` 是可信赖的，一般情况下很可能得出全局最优解。如果将 N 改为50，则总耗时降至12.408 s，全局最优解成功率为44%。

从这里给出的搜索算法看，应该很适合使用并行计算的策略加快搜索过程，不过若尝试使用并行计算，则MATLAB执行机构会给出错误信息，指出匿名函数句柄的处理不能进行并行资源的分派，从而不能使用并行计算的方法求解问题。

3.4 带有决策变量边界的最优化问题

本章前面的内容介绍了理论上的无约束最优化问题，但在很多演示例子中，采用的是指定决策变量范围的“无约束”最优化问题。例如，例3-2给出的函数在不同感兴趣区间的全局最优解是不同的，都不是数学意义下的无约束最优化问题，所以，本节将考虑带有决策变量边界的最优化问题的求解方法，首先介绍单变量最优化求解问题，然后给出多变量最优化的求解方法。

3.4.1 单变量最优化问题

本节先给出带有决策变量边界的单变量无约束最优化问题的定义，然后介绍MATLAB提供的直接求解函数，再介绍全局优化的解决方法。

定义 3-4 ► 带边界的最优化问题

带有决策变量边界的单变量最优化问题的数学形式为

$$\min_{x\ \text{s.t.}\ x_m\leqslant x\leqslant x_M} f(x) \tag{3-4-1}$$

式中，x_m 与 x_M 分别为决策变量的下界与上界，决策变量 x 为标量。

式（3-4-1）中，记号 s.t. 是英文 subject to 的缩写，表示满足后面的关系。

MATLAB 最优化工具箱提供了 fminbnd() 函数，可以用于直接求解带有决策变量的单变量最优化问题，该函数的调用格式为

x=fminbnd(f,x_{m},x_{M})

[x,f_0,flag,out]=fminbnd(f,x_{m},x_{M},options)

[x,f_0,flag,out]=fminbnd(problem)

式中，f 是单变量目标函数的句柄；x_{m} 和 x_{M} 分别为决策变量的下界与上界。如果采用结构体 problem 描述最优化问题，则 x_{m} 和 x_{M} 相应的成员变量名分别为 x1 和 x2。注意，fminbnd() 函数不支持 optimoptions() 函数设置的控制选项，只能由 optimset() 函数设置。

例 3-29　试重新求解例 3-3 中的最优化问题。

解　显然，这个最优化问题带有决策变量的边界 $t_{\mathrm{m}}=-0.5$，$t_{\mathrm{M}}=2.5$，可以由下面的命令直接求解最优化问题，得出的结果为 $t^*=1.5511$。遗憾的是，这样的求解函数并不能得出感兴趣区域的全局最优解，即图 3-3 中标注的 t_0，而是局部最优解 t_3，而 fminbnd() 函数本身并不能选择初值，所以，只利用该函数并不能有效地求解全局最优化问题。

```
>> f=@(t)exp(-2*t).*cos(10*t)+exp(-3*(t+2)).*sin(2*t);
   tm=-0.5; tM=2.5; t=fminbnd(f,tm,tM)
```

还可以使用结构体描述原始问题，由下面的语句求解问题，得出完全一致的结果。

```
>> clear P                          %清除现有的P变量
   P.objective=@(t)exp(-2*t).*cos(10*t)+exp(-3*(t+2)).*sin(2*t);
   P.options=optimset; P.solver='fminbnd';
   P.x1=-0.5; P.x2=2.5;             %决策变量边界
   t=fminbnd(P)                     %求解最优化问题
```

由于该函数本身没有可调的参数，所以为得到全局最优解，需要将 $(x_{\mathrm{m}},x_{\mathrm{M}})$ 区间划分成 m 个子区间，对每个子区间都求取最小值，找出其中目标函数值最小的一个，这个值就很可能是问题的全局最优解。利用这样的思路可以编写出下面的 MATLAB 函数，该程序中使用的 fminsearchbnd() 函数后面将介绍。

```
function [x,f0]=fminbnd_global(f,xm,xM,n,m,varargin)
   arguments, f, xm(:,1), xM(:,1)
      n(1,1) {mustBePositiveInteger}
      m(1,1) {mustBePositiveInteger}
   end
   argumemts (Repeating), varargin; end
   f0=Inf; M=ones(n,1);
```

```
    if isscalar(xm), xm=xm*M; end
    if isscalar(xM), xM=xM*M; end
    for i=1:m
       if n==1, h=(xM-xm)/m;                    %单变量问题
          [x1,f1]=fminbnd(f,xm+(i-1)*h,xm+i*h,varargin{:});
       else, x0=xm+(xM-xm).*rand(n,1);    %多变量问题
          [x1 f1 key]=fminsearchbnd(f,x0,xm,xM,varargin{:});
       end                                  %用循环结构生成随机初始搜索点
       if f1<f0, x=x1; f0=f1; end    %如果得到更好的解,则更新记录
end, end
```

该函数的调用格式为 [x,f_0]=fminbnd_global(f,x_m,x_M,1,m)，其中，应该为 m 取一个较大的值，这样得出的解 x 很可能是原始问题的全局最优解。

例 3-30 试重新求解例 3-2 中的最优化问题，并得出全局最优解。

解 取子区间个数为 $m=10$，则可以调用新的求解函数计算原问题的全局最优解，得出的解为 $x=-0.3340$，确实能得到图 3-3 中标注的全局最优解 t_0。

```
>> f=@(t)exp(-2*t).*cos(10*t)+exp(-3*(t+2)).*sin(2*t);
   tm=-0.5; tM=2.5; x=fminbnd_global(f,tm,xM,1,10)
```

3.4.2 多变量最优化问题

定义 3-5 ► 多变量最优化问题

如果定义 3-4 中的 x 为 $\boldsymbol{x}$ 向量，则相应的最优化问题可以改写成

$$\min_{\boldsymbol{x}\ \text{s.t.}\ \boldsymbol{x}_\mathrm{m}\leqslant\boldsymbol{x}\leqslant\boldsymbol{x}_\mathrm{M}} f(\boldsymbol{x}) \tag{3-4-2}$$

此时的最优化问题称为带有决策变量边界的多变量最优化问题。

式（3-4-2）描述问题的物理意义是，$\boldsymbol{x}$ 在指定的范围内取多少时能使得目标函数取最优值，这样的问题由 fminsearch() 和 fminbnd() 函数是不能直接求解的。

John D'Errico 提出了一种变换方法[12]，引入新决策变量 z_i，使得

$$x_i=x_{\mathrm{m}_i}+\frac{1}{2}(x_{\mathrm{M}_i}-x_{\mathrm{m}_i})(\sin z_i+1) \tag{3-4-3}$$

这样，就可以将关于 x_i 的决策变量边界问题巧妙地转换成关于 z_i 的无约束最优化问题。John D'Errico 还开发了 fminsearchbnd() 函数，扩展了现有 fminsearch() 函数的功能，使其能直接求解式（3-4-2）中的问题。fminsearchbnd() 函数的调用格式为

[$\boldsymbol{x}$,f_0,flag,out]=fminsearchbnd(f,$\boldsymbol{x}_0$,$\boldsymbol{x}_\mathrm{m}$,$\boldsymbol{x}_\mathrm{M}$)

[$\boldsymbol{x}$,f_0,flag,out]=fminsearchbnd(f,$\boldsymbol{x}_0$,$\boldsymbol{x}_\mathrm{m}$,$\boldsymbol{x}_\mathrm{M}$,opt)

[$\boldsymbol{x}$,f_0,flag,out]=fminsearchbnd(f,$\boldsymbol{x}_0$,$\boldsymbol{x}_{\mathrm{m}}$,$\boldsymbol{x}_{\mathrm{M}}$,opt,$p_1$,$p_2$,$\cdots$)

如果没有给出上界或下界约束，则可以将其设置为空矩阵 []，这时，该函数会自动调整式（3-4-3）中的变换公式，仍然能正常求解最优化问题。

例 3-31　试求解下面函数的最小值。

$$f(x,y)=(1.5-x+xy)^2+\left(2.25-x+xy^2\right)^2+\left(2.625-x+xy^3\right)^2$$

其中，$-4.5\leqslant x,y\leqslant 4.5$。

解　令 $x_1=x,x_2=y$，可以将目标函数改写成

$$f(\boldsymbol{x})=(1.5-x_1+x_1x_2)^2+\left(2.25-x_1+x_1x_2^2\right)^2+\left(2.625-x_1+x_1x_2^3\right)^2$$

可以由匿名函数的形式直接由点运算描述目标函数，然后绘制出目标函数的曲面，如图 3-16 所示。

```
>> f=@(x,y)(1.5-x+x.*y).^2+(2.25-x+x.*y.^2).^2+...
          (2.625-x+x.*y.^3).^2;
   fsurf(f,[-4.5,4.5])
```

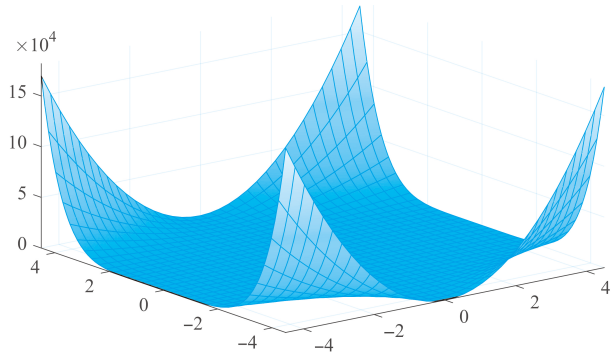

图 3-16　目标函数的曲面

如果想求解原始的最优化问题，并没有必要真采用上面 $f(\boldsymbol{x})$ 的格式重新输入目标函数，只需按照下面的形式改写目标函数，就可以构造出目标函数句柄 f_0。这时再求解最优化问题，就可以得出全局最优解为 $\boldsymbol{x}_1=[3,0.5]^{\mathrm{T}}$，这时的目标函数为 $f(\boldsymbol{x}_1)=2.6871\times10^{-29}$。

```
>> f0=@(x)f(x(1),x(2));
   opt=optimset; opt.TolX=eps; ff.TolFun=eps;
   [x1,f1]=fminsearchbnd(f0,[0;0],-4.5*[1;1],4.5*[1;1],opt)
```

如果 $-2\leqslant x,y\leqslant 2$，则显然刚才得到的最优解不在此区域内，所以可以利用下面的语句重新搜索，得出的最优解为 $\boldsymbol{x}_1=[2,0.1701]^{\mathrm{T}}$，这时的最优目标函数值为 $f_1=0.5233$。

```
>> [x1,f1]=fminsearchbnd(f0,[0;0],-2*[1;1],2*[1;1],opt)
   fsurf(f,[-2,2])     % 目标函数曲面
```

3.4.3 基于问题的描述与求解

前面介绍了 optimvar() 函数的调用格式，可以用该函数定义决策变量，事实上，该变量还可以由下面的语句直接定义。

$\boldsymbol{x}$=optimvar('x',[n,m,k],'LowerBound',$\boldsymbol{x}_{\mathrm{m}}$,'UpperBound',$\boldsymbol{x}_{\mathrm{M}}$) 或

$\boldsymbol{x}$=optimvar('x',[n,m,k],LowerBound=$\boldsymbol{x}_{\mathrm{m}}$,UpperBound=$\boldsymbol{x}_{\mathrm{M}}$)

其中，n、m 和 k 为三维数组的维数；如果不给出 k，则可以定义出 $n \times m$ 决策矩阵 $\boldsymbol{x}$；若 m 为 1，则可以定义 $n \times 1$ 决策列向量。如果 $\boldsymbol{x}_{\mathrm{m}}$ 为标量，则可以将全部决策变量的下限都设置成相同的值。属性名 LowerBound 可以简化成 Lower。也可以用类似的方法定义 UpperBound 属性，简称 Upper。

用这样的方法定义决策变量，就可以直接限制其上限与下限，再调用 solve() 函数，直接求解带有决策变量边界的无约束最优化问题。

例 3-32 试用基于问题的求解方法求解例 3-31 中的问题。方便起见，这里重新给出了原始的目标函数。

$$f(x,y)=(1.5-x+xy)^2+\left(2.25-x+xy^2\right)^2+\left(2.625-x+xy^3\right)^2$$

其中，$-4.5 \leqslant x,y \leqslant 4.5$。

解 定义两个决策变量，并指定其上下界，就可以直接用基于问题的方法求解最优化问题，得出的结果为 $x=3, y=0.5$，其目标函数为 2.5397×10^{-12}，与例 3-31 的结果完全一致。

```
>> P=optimproblem;                                    %创建最优化模型
   x=optimvar('x',1,Lower=-4.5,Upper=4.5);            %定义决策变量
   y=optimvar('y',1,Lower=-4.5,Upper=4.5);            %定义决策变量
   P.Objective=(1.5-x+x*y)^2+(2.25-x+x*y^2)^2+(2.625-x+x*y^3)^2;
   x0.x=10*rand(1,1); x0.y=10*rand(1,1);              %设定初值
   [sol,f0,flag]=solve(P,x0), sol.x, sol.y            %直接求解最优化问题
```

3.4.4 边界问题全局最优解的尝试

对多峰的目标函数或大范围搜索，前面介绍的 fminsearchbnd() 函数可能不能确保得出问题的全局最优解，所以需要引入全局最优的方法。可以尝试前面编写的函数 fminbnd_global() 求取最优化问题的全局最优解。

例 3-33 如果 $-500 \leqslant x,y \leqslant 500$，试求解例 3-31 中的最优化问题。

解 如果使用 fminsearchbnd() 函数，当然可以给出下面的语句，不过得出的结果可能为 $\boldsymbol{x}_1=[-50,1.0195]^{\mathrm{T}}$，且 $f_0=0.4823$，同时系统给出警告信息，“超过了函数计算的最大数目，请增大 MaxFunEvals 选项”，说明求解不成功。反复运行下面最后一条指令可以发现，大约有一半的运行得不出全局最优解 $[3,0.5]$。

```
>> f=@(x,y)(1.5-x+x.*y).^2+...
          (2.25-x+x.*y.^2).^2+(2.625-x+x.*y.^3).^2;
   f0=@(x)f(x(1),x(2));
   ff=optimset; ff.TolX=eps; ff.TolFun=eps;
   [x1,f1]=fminsearchbnd(f0,500*rand(2,1),-500*[1;1],500*[1;1],ff)
```

调用前面编写的 fminbnd_global() 函数重新求解原始问题，反复运行下面的命令，可以看出 100% 的运行都能得到原始问题的全局最优解。

```
>> [x1,f1]=fminbnd_global(f0,-500,500,2,5,ff)
```

3.5　最优化问题应用举例

本节将给出几个最优化技术应用的实例。首先引入线性回归问题的求解方法，如果已知实验数据，则通过实验数据将其对应的数学模型重建出来。这类方法对应于超定线性方程的最小二乘求解方法。如果函数的数学模型的形式过于复杂，本节还将介绍基于最小二乘方法的曲线拟合技术。另外，本节还将通过例子演示基于最优化技术的微分方程边值问题与代数方程的求解方法。

其实，本节给出的几个例子是想演示一下，如何将看起来不相关的问题转换成最优化问题，通过最优化问题的建模与求解，得出原始问题的解。

3.5.1　线性回归问题的求解

假设已知某函数的线性组合为

$$g(x)=c_1f_1(x)+c_2f_2(x)+c_3f_3(x)+\cdots+c_nf_n(x) \tag{3-5-1}$$

式中，$f_1(x),f_2(x),\cdots,f_n(x)$ 为已知函数；$c_1,c_2,\cdots,c_n$ 为待定系数。这时假设已经测出数据 $(x_1,y_1),(x_2,y_2),\cdots,(x_m,y_m)$，则可以建立如下线性方程：

$$\boldsymbol{Ac}=\boldsymbol{y} \tag{3-5-2}$$

式中，

$$\boldsymbol{A}=\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix},\quad \boldsymbol{y}=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{3-5-3}$$

且 $\boldsymbol{c}=[c_1,c_2,\cdots,c_n]^{\mathrm{T}}$，故该方程的最小二乘解为 **c=A\y**。

例 3-34　假设测出了一组 (x_i,y_i)，由表 3-4 给出，且已知函数原型如下，试用已知的数据求出待定系数 c_i 的值。

$$y(x)=c_1+c_2\mathrm{e}^{-3x}+c_3\cos(-2x)\mathrm{e}^{-4x}+c_4x^2$$

表 3-4 例 3-34 实测数据

x_i	0	0.2	0.4	0.7	0.9	0.92	0.99	1.2	1.4	1.48	1.5
y_i	2.88	2.2576	1.9683	1.9258	2.0862	2.109	2.1979	2.5409	2.9627	3.155	3.2052

解 可以由表 3-4 中给出的数据直接拟合出曲线方程中的 c_i 参数。这样，就可以依照各个子函数的表达式，由点运算的方式构造 $\boldsymbol{A}$ 矩阵，再使用最小二乘命令求出各个待定系数。

```
>> x=[0,0.2,0.4,0.7,0.9,0.92,0.99,1.2,1.4,1.48,1.5]'; %样本点输入
   y=[2.88;2.2576;1.9683;1.9258;2.0862;2.109;2.1979;
      2.5409;2.9627;3.155;3.2052];
   A=[ones(size(x)) exp(-3*x),cos(-2*x).*exp(-4*x) x.^2];
   c=A\y; c1=c'                                          %最小二乘
```

可以得出拟合参数 $\boldsymbol{c}^{\mathrm{T}}=[1.22, 2.3397, -0.6797, 0.87]$，将更密集的 $\boldsymbol{x}$ 向量代入该原型函数，拟合曲线和已知数据点如图 3-17 所示，可见拟合效果是令人满意的。

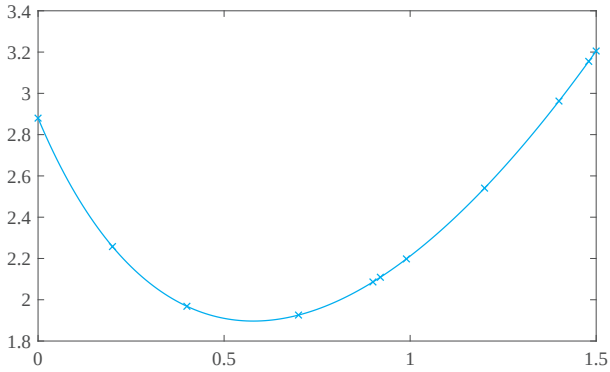

图 3-17 原始数据与拟合曲线

```
>> x0=[0:0.01:1.5]';
   B=[ones(size(x0)) exp(-3*x0) cos(-2*x0).*exp(-4*x0) x0.^2];
   y1=B*c; plot(x0,y1,x,y,'x') %拟合曲线计算与绘制
```

3.5.2 曲线的最小二乘拟合

前面介绍的线性回归方法中，首先假定 $f_i(x)$ 为已知函数，所以待定系数的计算才能转换成线性代数方程的求解问题。在实际应用中，$f_i(x)$ 这类函数本身可能带有其他待定系数，这时这类问题就不能由线性代数方程求解，必须将其转换为最优化问题进行求解。

定义 3-6 ▶ 最小二乘

假设有一组数据 $x_i, y_i, i = 1, 2, \cdots, m$，且已知这组数据满足某一函数原型 $\hat{y}(x) = f(\boldsymbol{a}, x)$，其中，$\boldsymbol{a}$ 为待定系数向量，则最小二乘曲线拟合的目标就是求出这一组待定系数的值，可以将问题转换成下面的最优化问题：

$$J = \min_{\boldsymbol{a}} \sum_{i=1}^{m} \left[y_i - \hat{y}(x_i) \right]^2 = \min_{\boldsymbol{a}} \sum_{i=1}^{m} \left[y_i - f(\boldsymbol{a}, x_i) \right]^2 \tag{3-5-4}$$

该最优化问题可以由底层求解的方式求解，也可以调用 MATLAB 的最优化工具箱中提供的 `lsqcurvefit()` 函数，该函数可以解决最小二乘曲线拟合的问题。该函数的调用格式为

`[`$\boldsymbol{a}$`,`J_{m}`,cc,flag,out]=lsqcurvefit(Fun,`$\boldsymbol{a}_0$`,`$\boldsymbol{x}$`,`$\boldsymbol{y}$`,options)`

式中，`Fun` 为原型函数的 MATLAB 表示，可以是 M 函数或匿名函数；$\boldsymbol{a}_0$ 为最优化的初值；$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 分别为原始输入和输出数据向量，对多元函数的拟合而言，这两个输入变元还可以是矩阵，后面将通过例子演示这种情况；`options` 则为最优化工具箱通用的控制模板。调用该函数将返回待定系数向量 $\boldsymbol{a}$，以及在此待定系数下的目标函数的值 J_{m}。如果 `flag` 的值为正，则说明求解成功。

例 3-35　假设由下面的语句生成一组数据 $\boldsymbol{x}$ 和 $\boldsymbol{y}$。

```
>> x=0:0.4:10;                    %生成样本点数据
   y=0.12*exp(-0.213*x)+0.54*exp(-0.17*x).*sin(1.23*x);
```

并已知该数据满足原型函数为 $y(x) = a_1 \mathrm{e}^{-a_2 x} + a_3 \mathrm{e}^{-a_4 x} \sin(a_5 x)$，其中，$a_i$ 为待定系数。采用最小二乘曲线拟合的目的就是获得这些待定系数，使得目标函数的值为最小。

解　显然，这类问题不能由前面介绍的线性回归方法直接求解，只能采用这里介绍的最小二乘曲线拟合方法求解。

根据已知的函数原型，可以编写出如下匿名函数。建立起函数的原型，即可以由下面的语句得出待定系数向量为 $\boldsymbol{c} = [0.12, 0.213, 0.54, 0.17, 1.23]$，拟合残差为 1.7928×10^{-16}。可以看出，得出的待定系数精度较高。下面的语句还可以绘制出拟合曲线与样本点，如图 3-18 所示，可见拟合精度很高。

```
>> f=@(a,x)a(1)*exp(-a(2)*x)+a(3)*exp(-a(4)*x).*sin(a(5)*x);
   a0=[1,1,1,1,1]; [xx,res]=lsqcurvefit(f,a0,x,y); %最小二乘拟合
   x1=0:0.01:10; y1=f(xx,x1); plot(x1,y1,x,y,'o')  %拟合效果比较
```

其实，由底层命令求解该最优化问题也不困难，只需依赖前面给出的匿名函数，定义一个新的匿名函数描述目标函数，就可以由下面的语句直接求解出所需的待定系数，这样得出的结果与前面的结果完全一致。

```
>> F=@(a)norm(f(a,x)-y); x1=fminunc(F,a0)
```

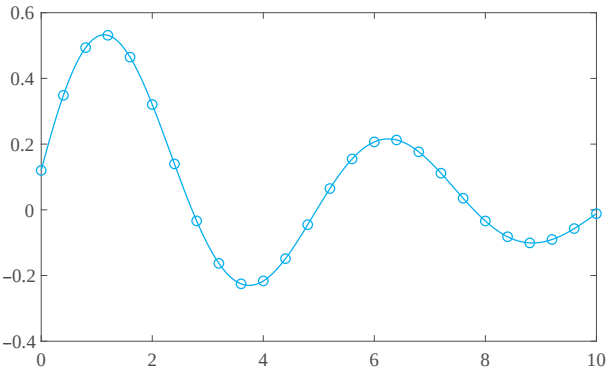

图 3-18 拟合效果比较

例 3-36 假设有一组实测数据由表 3-5 给出，且已知该数据可能满足的原型函数为 $y(x)=ax+bx^2\mathrm{e}^{-cx}+d$，试求出满足下面数据的最小二乘解 a、b、c 和 d 的值。

表 3-5 例 3-36 实测数据

x_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y_i	2.3201	2.6470	2.9707	3.2885	3.6008	3.9090	4.2147	4.5191	4.8232	5.1275

解 表 3-5 给出的样本点数据可以通过下面的语句直接输入 MATLAB 工作空间。

```
>> x=0.1:0.1:1;      %输入样本点数据
   y=[2.3201,2.6470,2.9707,3.2885,3.6008,3.9090,...
      4.2147,4.5191,4.8232,5.1275];
```

令 $a_1=a$，$a_2=b$，$a_3=c$，$a_4=d$，这样，原型函数可以写成 $y(x)=a_1x+a_2x^2\mathrm{e}^{-a_3x}+a_4$，可以用匿名函数描述该原型函数。下面的语句可以得出函数的待定参数 $\boldsymbol{a}=[3.1001,1.5027,4.0046,2]^{\mathrm{T}}$。注意，本例若不采用循环，则可能收敛不到真值。

```
>> f=@(a,x)a(1)*x+a(2)*x.^2.*exp(-a(3)*x)+a(4);
   a=[1;2;2;3];    %原型函数与初值
   while (1)
      [a,f0,cc,flag]=lsqcurvefit(f,a,x,y);
      if flag>0, break;
   end, end         %最小二乘
```

用下面的语句还可以计算出各个点处的值，可以将拟合曲线与样本点曲线绘制在同一坐标系下，如图 3-19 所示。可见，二者还是很接近的，说明拟合效果较好。

```
>> y1=f(a,x); plot(x,y,x,y1,'o')      %拟合效果比较
```

如果某函数含有若干自变量，且已知其原型函数 $\boldsymbol{z}=f(\boldsymbol{a},x_1,\ x_2,\cdots,x_m)$，则仍然可以使用 `lsqcurvefit()` 函数拟合参数 $\boldsymbol{a}$，其中，$\boldsymbol{a}=[a_1,a_2,\cdots,a_n]$，该函数仍需要用户编写一个匿名函数或 MATLAB 函数描述原型函数，然后调用函数

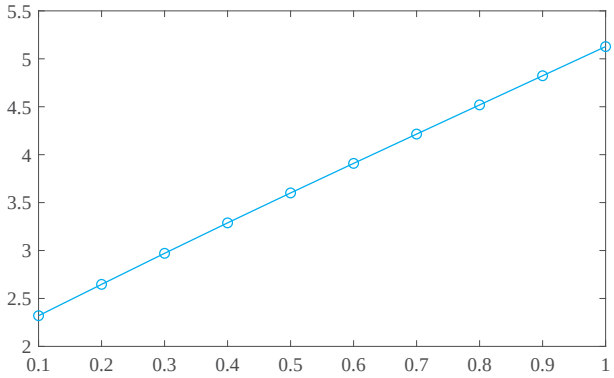

图 3-19　拟合效果比较

lsqcurvefit() 直接求解待定系数向量 $\boldsymbol{a}$。下面将通过例子演示多变量函数最小二乘拟合的求解方法。

例 3-37　假设某三元函数的原型函数为

$$v = a_1 x^{a_2 x} + a_3 y^{a_4(x+y)} + a_5 z^{a_6(x+y+z)}$$

且已知一组输入和输出数据，由文本文件 c3data1.dat 给出，该文件的前三列为自变量 x、y 和 z，第四列为返回向量，试采用拟合方法得出待定系数 a_i。

解　解决这类问题第一步仍然需要引入向量型的自变量 $\boldsymbol{x}$，如令 $x_1 = x$，$x_2 = y$，$x_3 = z$，这样，原型函数可以重新表示为

$$v = a_1 x_1^{a_2 x_1} + a_3 x_2^{a_4(x_1+x_2)} + a_5 x_3^{a_6(x_1+x_2+x_3)}$$

因为给出的数据是纯文本文件，故可以通过 load() 函数将其读入 MATLAB 工作空间。用子矩阵提取的方法将输入矩阵 $\boldsymbol{X}$ 和输出向量 $\boldsymbol{v}$ 提取出来，可以用下面的语句拟合出待定系数的值 $\boldsymbol{a} = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$，使得拟合误差的最小平方和最小，其值为 1.0904×10^{-7}。注意，在匿名函数使用第 i 个自变量时，一定要给出 $\boldsymbol{X}(:,i)$ 命令提取该自变量。

```
>> f=@(a,X)a(1)*X(:,1).^(a(2)*X(:,1))+...
          a(3)*X(:,2).^(a(4)*(X(:,1)+X(:,2)))+...
          a(5)*X(:,3).^(a(6)*(X(:,1)+X(:,2)+X(:,3))); %原型函数
   XX=load('c3data1.dat'); X=XX(:,1:3); v=XX(:,4); %读入样本点数据
   a0=[2 3 2 1 2 3]; [a,f,err,flag]=lsqcurvefit(f,a0,X,v) %最小二乘
```

事实上，文件中给出的数据就是假设 $\boldsymbol{a} = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]$ 生成的，所以用这里给出的拟合方法可以很精确地得出待定系数。

3.5.3　边值微分方程的打靶求解

本书卷 V 将全面介绍微分方程的解析解与数值解，这里只给出一个简单的例子演示边值微分方程问题的打靶求解方法，并介绍如何将这类方法转换成最优化

问题，得出问题的解。

对一般的微分方程而言，需要求解的是初值问题，即已知微分方程的数学描述

$$\boldsymbol{x}'(t)=\boldsymbol{f}(t,\boldsymbol{x}) \tag{3-5-5}$$

且已知 $\boldsymbol{x}(t_0)$，这样，该方程可以调用 ode45() 函数直接求出数值解。

在实际应用中，如果已知 $\boldsymbol{x}(t_0)$ 中的几个分量，同时还已知 $\boldsymbol{x}(t_n)$ 的另外几个值，如何求这里微分方程的数值解呢？打靶方法是常用的求解方法。

打靶法的思路是这样的：先给 $\boldsymbol{x}(t_0)$ 的另外几个值赋初值，就可以认为微分方程的初值为已知的，所以可以调用 ode45() 求解微分方程，得出终点处微分方程的数值解 $\hat{\boldsymbol{x}}(t_n)$。这样，$\hat{\boldsymbol{x}}(t_n)$ 与事先给定 $\boldsymbol{x}(t_n)$ 中的几个已知的量相比，就存在误差量了，根据该误差量就可以调整 $\boldsymbol{x}(t_0)$ 的初值，重新求解微分方程。采用这样的方法就可以构造一个循环，对 $\hat{\boldsymbol{x}}(t_n)$ 与 $\boldsymbol{x}(t_n)$ 中的几个已知的量的误差的范数之和作为最优化的目标函数，从而得出相容的初值，最终得出微分方程边值问题的数值解。

例 3-38 假设已知微分方程：

$$\begin{cases} x_1'(t)=x_2(t) \\ x_2'(t)=-x_1(t)-3x_2(t)+\mathrm{e}^{-5t} \\ x_3'(t)=x_4(t) \\ x_4'(t)=2x_1(t)-4x_2(t)-3x_3(t)-4x_4(t)-\sin t \end{cases}$$

且已知边值 $x_1(0)=1, x_2(0)=2, x_3(10)=-0.021677, x_4(10)=0.15797$，试求解该微分方程在 $t\in(0,10)$ 时的数值解。

解 由于已知初值 $x_1(0)$ 和 $x_2(0)$，所以可以保留这两个给定的值，而将未知的 $x_3(0)$ 和 $x_4(0)$ 选作决策变量，这样原始问题就可以转换成最优化问题

$$\min_{x_3(0),x_4(0)} \quad ||x_3(10)-\hat{x}_3(10)||+||x_4(10)-\hat{x}_4(10)||$$

令 $y_1=x_3(0), y_2=x_4(0)$，则最优化问题的数学形式可以改写成

$$\min_{\boldsymbol{y}} \ ||y_1-\hat{x}_3(10)||+||y_2-\hat{x}_4(10)||$$

而 $\hat{x}_3(10)$ 和 $\hat{x}_4(10)$ 是以 $[x_1(0),x_2(0),y_1,y_2]^{\mathrm{T}}$ 为初值的微分方程解的后两项终值，所以目标函数可以如下描述。注意，这里只能采用 M 函数描述微分方程，不能采用匿名函数，因为该函数的内部是有中间变量的。另外，f、$\boldsymbol{x}_0$ 与 $\boldsymbol{x}_n$ 是附加参数，后面将演示不采用附加参数的函数格式。

```
function z=c3mode(y,f,x0,xn)
   arguments, y(:,1), f, x0(:,1), xn(:,1); end
   x0=[x0(1:2); y]; [t,x]=ode45(f,[0,10],x0);
   z=norm(x(end,3:4)'-xn);
end
```

现在可以用匿名函数表示微分方程模型。将目标函数转换成不带有附加参数的匿名函数，可以由下面语句的直接求解等效的 $x_3(0)$ 与 $x_4(0)$，得出 $x_3(0)=0.10158462116\,3685$，$x_4(0)=-2.318606769109767$。在等效初值下求解微分方程，可以绘制出微分方程解的曲线，如图 3-20 所示。另外可以看出，得出微分方程的终值与给定的初值是完全一致的，说明求解过程是成功的。

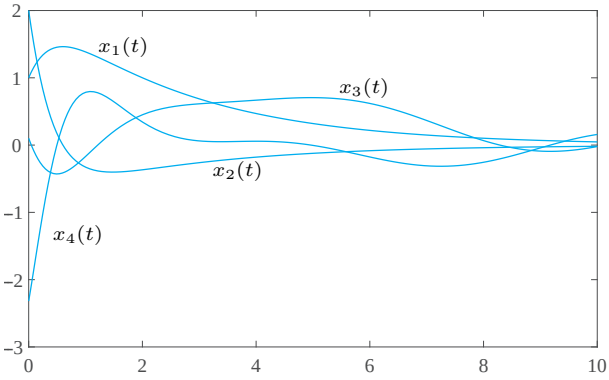

图 3-20　微分方程数值解曲线

```
>> f=@(t,x)[x(2); -x(1)-3*x(2)+exp(-5*t);
            x(4); 2*x(1)-3*x(2)-3*x(3)-4*x(4)-sin(t)];
   x0=[1; 2]; xn=[-0.021677; 0.15797];
   g=@(x)c3mode(x,f,x0,xn); %将带有附加参数的函数转换成普通匿名函数
   x2=rand(2,1); x3=fminunc(g,x2) %选择随机初值求解最优化问题
   [t,x]=ode45(f,[0,10],[x0; x3]); plot(t,x), x(end,:)
```

事实上，这里反推初值的方法得出的解可能是不唯一的。例如，如果初值向量选择为 $x_3(0)=-1$，$x_4(0)=1$，则得出的微分方程解的终值也满足给定数值的终值条件，新初始条件下微分方程的解如图 3-21 所示。在前面的求解语句中，每次将 $\boldsymbol{x}_2$ 设置成不同的随机数向量，得出的相容初值都可能是不同的，由相容初值求解微分方程都得到满足给定的终值条件。

```
>> [t,x]=ode45(f,[0,10],[x0; -1; 1]); x(end,:)
```

其实，除了将原始问题转换成最优化问题之外，还可以将问题转换为代数方程的数值求解问题。例如，对上面的问题还可以建立下面的代数联立方程：

$$\begin{cases} y_1-\hat{x}_3(10)=0 \\ y_2-\hat{x}_4(10)=0 \end{cases}$$

并通过方程数值求解的方法求出相容的 $x_3(0)$ 和 $x_4(0)$，从而求解微分方程的边值问题。下面通过例子演示基于代数方程求解的微分方程边值问题求解方法。

例 3-39　重新考虑例 3-38 中的微分方程求解问题，试用解方程的方法求解微分方

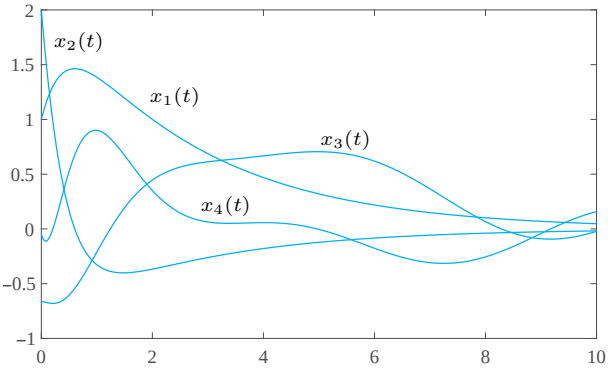

图 3-21　另一组微分方程数值解曲线

程的边值问题。

解　仿照前面的 c3mode() 函数，可以编写下面的 MATLAB 函数，描述联立方程。

```
function z=c3mode2(y,f,x0,xn)
   arguments, y(:,1), f, x0(:,1), xn(:,1); end
   x0=[x0(1:2); y]; [t,x]=ode45(f,[0,10],x0);
   z=x(end,3:4)'-xn;
end
```

由下面的语句求解代数方程，可以得出相容的初值，最终可以求解微分方程的边值问题。根据不同的随机初值，可能得出例 3-38 中微分方程的两个解。

```
>> f=@(t,x)[x(2); -x(1)-3*x(2)+exp(-5*t);
            x(4); 2*x(1)-3*x(2)-3*x(3)-4*x(4)-sin(t)];
   x0=[1; 2]; xn=[-0.021677; 0.15797];
   ff=optimoptions('fsolve'); ff.TolX=eps; ff.TolFun=eps;
   g=@(x)c3mode2(x,f,x0,xn);          %转换成普通匿名函数
   x2=rand(2,1); x3=fsolve(g,x2,ff)  %选择随机初值求解方程
   [t,x]=ode45(f,[0,10],[x0; x3]); plot(t,x), x(end,:)
```

3.5.4　方程求解问题转换为最优化问题

第 2 章已经介绍了很多非线性方程求解的知识。其实，可以很容易地将方程求解问题转换为无约束最优化问题。假设想求解方程 $\boldsymbol{f}(\boldsymbol{x})=\boldsymbol{0}$，那么方程的解 $\boldsymbol{x}$ 会满足什么条件呢？当然，方程的解满足使得方程等于 $f_i(\boldsymbol{x})=0$ 这样的函数条件，所以可以考虑将这些函数的平方和作为目标函数进行寻优计算，找到决策变量 $\boldsymbol{x}$。这样，可以写出下面的最优化问题：

$$\min_{\boldsymbol{x}} \ f_1^2(\boldsymbol{x})+f_2^2(\boldsymbol{x})+\cdots+f_n^2(\boldsymbol{x})=\min_{\boldsymbol{x}} \ \sum_{i=1}^{n} f_i^2(\boldsymbol{x}) \tag{3-5-6}$$

例 3-40　试求解下面给出的二元联立方程组[4]。

$$\begin{cases} 4x_1^3 + 4x_1x_2 + 2x_2^2 - 42x_1 = 14 \\ 4x_2^3 + 2x_1^2 + 4x_1x_2 - 26x_2 = 22 \end{cases}$$

解　可以看出，这里的方程是例 2-51 欠定方程的变换表示形式。很自然地，可以将方程求解的问题变成如下最优化问题：

$$\min_{\boldsymbol{x}} \quad (4x_1^3 + 4x_1x_2 + 2x_2^2 - 42x_1 - 14)^2 + (4x_2^3 + 2x_1^2 + 4x_1x_2 - 26x_2 - 22)^2$$

可以先用匿名函数分别定义出两个方程，绘制出两个方程的隐函数曲线，如图 3-22 所示。从得出的曲线可见，原方程有九个实根，当然用前面介绍的 `more_sols()` 函数可以立即得出全部的九个根。

```
>> f1=@(x1,x2)4*x1.^3+4*x1.*x2+2*x2.^2-42*x1-14;
   f2=@(x1,x2)4*x2.^3+2*x1.^2+4*x1.*x2-26*x2-22;
   fimplicit({f1,f2})
```

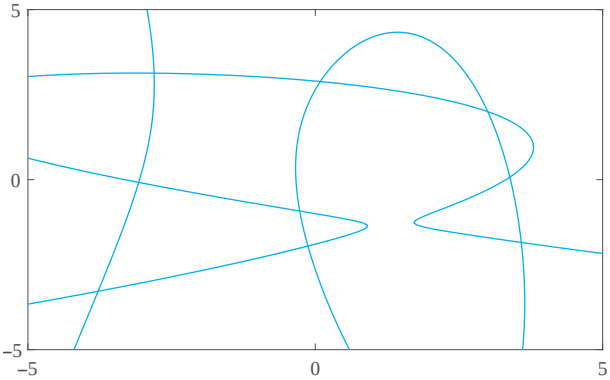

图 3-22　二元联立方程解的图示

现在考虑方程的最优化解法，借用前面定义的两个匿名函数构造出目标函数，然后选择初值求解，即可以得出方程的一个根。如果改变初值，则可能得出其他根。

```
>> f=@(x)(f1(x(1),x(2)))^2+(f2(x(1),x(2)))^2;
   [x,f0]=fminunc(f,rand(2,1));
```

从求解效率和精度看，该函数逊于专用于方程求解的 `fsolve()` 函数，这里给出这样的例子主要演示如何将一个一般问题转换为最优化问题，并利用最优化技术求解问题的思想。

从这里给出的例子可以看出，学习了最优化的思想，就可以有意识地在某些问题的研究中应用该思想。在实际应用中，比较关键的一步就是设计一个有物理意义的目标函数，并围绕该目标函数探讨最优化问题的求解方法。

本章习题

3.1 试求出使 $\int_0^1 (\mathrm{e}^x - cx)^2 \,\mathrm{d}x$ 取极小值的 c 值。

3.2 试求解下面的无约束最优化问题。

$$\min_{\boldsymbol{x}} \quad \begin{aligned} &100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2) + (1 - x_3^2)^2 + \\ &\quad 10.1\left[(x_2 - 1)^2 + (x_4 - 1)^2\right] + 19.8(x_2 - 1)(x_4 - 1) \end{aligned}$$

3.3 试找出下面二元函数曲面的全局谷底。

$$f(x_1, x_2) = -\frac{\sin\left(0.1 + \sqrt{(x_1 - 4)^2 + (x_2 - 9)^2}\right)}{1 + (x_1 - 4)^2 + (x_2 - 9)^2}$$

3.4 试求解 Griewangk 基准测试问题（$n = 20$）。

$$\min_{\boldsymbol{x}} \quad \left(1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\frac{x_i}{\sqrt{i}}\right), \quad x_i \in [-600, 600]$$

3.5 试求解 Ackley 基准测试问题[13]。

$$\min_{\boldsymbol{x}} \quad \left[20 + 10^{-20}\exp\left(-0.2\sqrt{\frac{1}{p}\sum_{i=1}^{p} x_i^2}\right) - \exp\left(\frac{1}{p}\sum_{i=1}^{p}\cos 2\pi x_i\right)\right]$$

3.6 试求解 Kursawe 基准测试问题。

$$J = \min_{\boldsymbol{x}} \quad \sum_{i=1}^{p} |x_i|^{0.8} + 5\sin^3 x_i + 3.5828$$

其中，可取 $p = 2$ 或 $p = 20$。

3.7 试求 Easom 函数的极小值。

$$f(\boldsymbol{x}) = -\cos x_1 \cos x_2 \mathrm{e}^{-(x_1 - \pi)^2 - (x_2 - \pi)^2}$$

其中，搜索范围为 $-10 \leqslant x_1, x_2 \leqslant 10$，问题的解析解为 $x_1 = \pi, x_2 = \pi$。

3.8 试求解扩展的 Freudenstein–Roth 函数的最小值问题（$n = 20$）。

$$f(x) = \sum_{i=1}^{n/2} \left[-13 + x_{2i-1} + ((5 - x_{2i})x_{2i} - 2)x_{2i}\right]^2 + \left[-29 + x_{2i-1} + ((x_{2i} + 1)x_{2i} - 14)x_{2i}\right]^2$$

初始点 $\boldsymbol{x}_0 = [0.5, -2, \cdots, 0.5, -2]^{\mathrm{T}}$，解析解 $\boldsymbol{x}^* = [5, 4, \cdots, 5, 4]^{\mathrm{T}}$，$f_{\mathrm{opt}} = 0$。如果搜索范围变大，一般求解方法还能否得出问题的全局最优解？

3.9 试求解下面的最优化问题[14]，并用图形法和解析解方法分别检验得出的结果。

$$\min_{x,y} f(x, y)$$

其中，

$$\begin{aligned} f(x, y) = &\left[1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\right] \times \\ &\left[30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)\right] \end{aligned}$$

3.10 试求解扩展三角函数的最小值问题($n=20$)。

$$f(x)=\sum_{i=1}^{n}\left[\left(n-\sum_{j=1}^{n}\cos x_j\right)+i(1-\cos x_i)-\sin x_i\right]^2$$

初始点 $\boldsymbol{x}_0=[1/n,1/n,\cdots,1/n]^{\mathrm{T}}$,解析解 $x_i=0$,$f_{\mathrm{opt}}=0$。

3.11 试求解扩展 Rosenbrock 函数的最小值问题($n=20$)。

$$f(x)=\sum_{i=1}^{n/2}100\left(x_{2i}-x_{2i-1}^2\right)^2+(1-x_{2i-1})^2$$

初始点 $\boldsymbol{x}_0=[-1.2,1,\cdots,-1.2,1]$,解析解 $x_i=1$,$f_{\mathrm{opt}}=0$。如果将 $x_{2i}-x_{2i-1}^2$ 替换成 $x_{2i}-x_{2i-1}^3$,则变成扩展 White–Holst 问题,试求解该问题。

3.12 试求解扩展 Beale 函数的最小值问题($n=20$)。

$$f(x)=\sum_{i=1}^{n/2}\left[1.5-x_{2i-1}(1-x_{2i})\right]^2+\left[2.25-x_{2i-1}(1-x_{2i}^2)^2\right]^2+\left[2.625-x_{2i-1}(1-x_{2i}^3)^2\right]^2$$

初始值 $\boldsymbol{x}_0=[1,0.8,\cdots,1,0.8]^{\mathrm{T}}$,解析解未知。

3.13 试求解两个 Raydan 函数的最小值问题($n=20$)。

$$f_1(x)=\sum_{i=1}^{n}\frac{i}{10}\left(\mathrm{e}^{x_i}-x_i\right),\quad f_2(x)=\sum_{i=1}^{n}\left(\mathrm{e}^{x_i}-x_i\right)$$

初始值 $\boldsymbol{x}_0=[1,1,\cdots,1]^{\mathrm{T}}$,解析解 $x_i=1$,$f_{1\mathrm{opt}}=\sum_{i=1}^{n}\frac{i}{10}$,$f_{2\mathrm{opt}}=n$。

3.14 试求解扩展 Miele–Cantrell 函数的最小值问题($n=20$)。

$$f(x)=\sum_{i=1}^{n/4}\left(\mathrm{e}^{4i-3}-x_{4i-2}\right)^2+100(x_{4i-2}-x_{4i-1})^6+\tan^4(x_{4i-1}-x_{4i})+x_{4i-3}^8$$

初始值 $\boldsymbol{x}_0=[1,2,2,2,\cdots,1,2,2,2]^{\mathrm{T}}$,解析解 $\boldsymbol{x}^*=[0,1,1,1,\cdots,0,1,1,1]^{\mathrm{T}}$,$f_{\mathrm{opt}}=0$。

3.15 上面很多习题都假设 $n=20$,如果 $n=200$,试重新求解这些习题。

3.16 对下面给出的 Hartmann 函数,试求解全局小值问题。

$$f(\boldsymbol{x})=-\sum_{i=1}^{4}\alpha_i\exp\left[-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2\right]$$

其中,$\boldsymbol{\alpha}=[1,1,2,3,3,2]^{\mathrm{T}}$,且

$$\boldsymbol{A}=\begin{bmatrix}3&10&30\\0.1&10&35\\3&10&30\\0.1&10&35\end{bmatrix},\quad \boldsymbol{P}=10^{-4}\times\begin{bmatrix}3689&1170&2673\\4699&4387&7470\\1091&8732&5547\\381.5&5743&8828\end{bmatrix}$$

3.17 试求解 Schwefel 函数的全局最小化问题($n=20$)。

$$f(\boldsymbol{x})=418.9829n-\sum_{i=1}^{n}x_i\sin\sqrt{|x_i|}$$

搜索范围为 $-500 \leqslant x_i \leqslant 500$，解析解 $x_i = 1$，$f_{\text{opt}} = 0$。并绘制出 $n = 2$ 时目标函数的曲面。

3.18 试求 Eggholder 函数的全局最小值。

$$f(x,y) = -(y+47)\sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\sin\sqrt{|x-(y+47)|}$$

并绘制出目标函数的曲面。

3.19 求解下面的最优化问题[15]。

$$\min_{\boldsymbol{x}} \sum_{i=1}^{10}\left[\ln^2(x_i-2)+\ln^2(10-x_i)\right] + \left(\prod_{i=1}^{10} x_i\right)^2$$

其中，$-2.001 \leqslant x_i \leqslant 9.999$，$i = 1, 2, \cdots, 10$。

3.20 试求解下面带有边界限制的最优化问题，其中目标函数为[4]

$$\begin{aligned} f(t) = {} & \frac{588600}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos(t-2\pi/3) - \cos^2\theta)r_0^2)^6} - \\ & \frac{1079.1}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos(t-2\pi/3) - \cos^2\theta)r_0^2)^3} + \\ & \frac{600800}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos t - \cos^2\theta)r_0^2)^6} - \\ & \frac{1071.5}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos t - \cos^2\theta)r_0^2)^3} + \\ & \frac{481300}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos(t+2\pi/3) - \cos^2\theta)r_0^2)^6} - \\ & \frac{1064.6}{(3r_0^2 - 4\cos\theta r_0^2 - 2(\sin^2\theta\cos(t+2\pi/3) - \cos^2\theta)r_0^2)^3} \end{aligned}$$

其中，$0 \leqslant t \leqslant 2\pi$，且常数 $r_0 = 1.54$，$\theta = 109.5°$。

3.21 试求全局最优化问题[14]。

$$\min_{x,y}\ 4x^2 - 2.1x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$$

3.22 试求出下面函数在 $-2 \leqslant x \leqslant 11$ 范围内的最小值，并用图形法检测结果。

$$f(x) = x^6 - \frac{52}{25}x^5 + \frac{39}{80}x^4 + \frac{71}{10}x^3 - \frac{79}{20}x^2 - x + \frac{1}{10}$$

3.23 试分别求出函数 $f(x) = x\sin 10\pi x + 2$ 在 $(-1, 2)$ 和 $(-10, 20)$ 区域下面函数的最大值，并绘制图形验证其结果。

3.24 试求出 $-10 \leqslant x, y \leqslant 10$ 时下面函数的最小值，并用图形显示这样的解。

$$f(x,y) = \sin^2 3\pi x + (x-1)^2\left(1+\sin^2 3\pi y\right) + (y-1)^2\left(1+\sin^2 2\pi y\right)$$

3.25 例 3-26 方程由许多波谷，如果仔细观察会发现谷底的函数值可能不同，试用图解法找出全局最优解（提示：考虑绘制侧视图与正视图，找出 x_1 和 x_2 的最优解）。

3.26 如果 $0 \leqslant x, y \leqslant 1$，试求出下面函数的最大值，并绘制图形验证结果。

$$f(x,y) = \sin 19\pi x + \frac{x}{1.7} + \sin 19\pi y + \frac{y}{1.7} + 2$$

3.27 试求 $-100 \leqslant x, y \leqslant 100$ 时下面函数的最小值。

$$f(x,y) = 0.5 + \frac{\cos^2\left(\sin|x^2 - y^2|\right) - 0.5}{\left[1 + 0.001(x^2 + y^2)\right]^2}$$

3.28 试求解下面的 Zakharov 函数测试问题。

$$\min \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i\right)^4$$

其中，$n = 20$，搜索范围为 $-5 \leqslant x_i \leqslant 10, i = 1, 2, \cdots, n$。

3.29 试求解下面的最优化问题。

$$f(\boldsymbol{x}) = 481.9829n + \sum_{j=1}^{n} \sin\sqrt{|x_j|}$$

其中，搜索范围为 $-500 \leqslant x_j \leqslant 500, n = 2$ 或 $n = 20$。

3.30 假设某一化学反应过程中压力 P 与温度 T 之间的关系为 $P = \alpha \mathrm{e}^{\beta T}$，但 α 与 β 是未知的。现通过实验测得一些数据，在表 3-6 中给出，试确定未知参数 α 与 β[16]。

表 3-6　习题3.30 中已知的实验数据

温度 T / °C	20	25	30	35	40	50	60	70
压力 P / mmHg	15.45	19.23	26.54	34.52	48.32	68.11	98.34	120.45

3.31 试求解下面函数的最小值。

$$f(\boldsymbol{x}) = \sum_{k=1}^{n} k\cos((k+1)x_k + k) \sum_{k=1}^{n} k\cos((k+1)x_k + k)$$

其中，$n = 20, -10 \leqslant x_k \leqslant 10$。

3.32 试求解下面的无约束最优化问题。

$$f(x,y) = \left[1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)\right] \times \left[30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)\right]$$

3.33 试求解下面的最优化问题。

$$f(\boldsymbol{x}) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x(4))^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2^- 1) * (x_4 - 1)$$

3.34 试求出下面函数的最小值。

$$f(x_1, x_2) = \sum_{j=1}^{n} \sin x_j \left(\sin j x_j^2/\pi\right)^m$$

可以尝试 $n = 2$ 和 $n = 20, m = 10$。

3.35 试求解下面带有决策变量约束的最优化问题。

$$\min_{x_1, x_2} \cos x_1 \sin x_2 - \frac{x_1}{1 + x_2^2}$$

其中，$-1 \leqslant x_1 \leqslant 2, -1 \leqslant x_2 \leqslant 1$。

3.36 假设已知一组数据如表 3-7 所示，且已知该数据满足原型函数。

$$y(x)=\frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-(x-\mu)^2/2\sigma^2}$$

试用最小二乘法求出 μ 和 σ 的值，并用得出的函数将函数曲线绘制出来，观察曲线拟合的效果。

表 3-7 习题3.36 数据

x_i	−2	−1.7	−1.4	−1.1	−0.8	−0.5	−0.2	0.1	0.4	0.7	1	1.3
y_i	0.1029	0.1174	0.1316	0.1448	0.1566	0.1662	0.1733	0.1775	0.1785	0.1764	0.1711	0.1630
x_i	1.6	1.9	2.2	2.5	2.8	3.1	3.4	3.7	4	4.3	4.6	4.9
y_i	0.1526	0.1402	0.1266	0.1122	0.0977	0.0835	0.0702	0.0577	0.0469	0.0373	0.0291	0.0224

3.37 假设有一组数据在文件 `data3ex5.dat` 中给出，其第一列为散点的 x 坐标，第二列为 y 坐标，第三列为函数值，试由文件的数据计算下面原型函数的待定系数 a~e。

$$f(x,y)=(ax^2-bx)\mathrm{e}^{-cx^2-dy^2-exy}$$

3.38 已知微分方程模型 $x_1'(t)=x_2(t)$，$x_2'(t)=2x_1(t)x_2(t)$，且已知边值条件 $x_1(0)=-1, x_1(\pi/2)=1$，试求解相应的微分方程数值解。

3.39 已知某常微分方程模型如下，试求出 α 和 β，并求解本微分方程。

$$x_1'=4x_1-\alpha x_1x_2,\ x_2'=-2x_2+\beta x_1x_2$$

且已知 $x_1(0)=2, x_2(0)=1, x_1(3)=4, x_2(3)=2$。

3.40 例 3-40 中给出了将方程求解问题转换为最优化问题的方法，试运行该函数 100 次，并运行 `fsolve()` 函数 100 次解决同样的问题，比较方程求解的效率与精度。

3.41 试将习题2.12、习题2.14 中给出的方程求解问题转换成最优化问题并求解方程，试比较方程求解的精度与速度。