第3章 粒子群算法及其MATLAB实现

粒子群算法，也称粒子群优化算法(particle swarm optimization，PSO)，是近年来发展起来的一种新的进化算法(evolutionary algorithm，EA)。

这种算法以其实现容易、精度高、收敛快等优点引起了学术界的重视，并且在解决实际问题中展示了其优越性。粒子群算法是一种并行算法。

本章主要讲解了粒子群算法的原理及其在MATLAB上的运用。

学习目标：

- 了解粒子群算法的发展。
- 掌握粒子群算法的基本原理。
- 熟悉MATLAB粒子群算法工具箱。
- 掌握MATLAB在粒子群算法中的运用。

3.1 粒子群算法基础

PSO算法属于进化算法的一种，和模拟退火算法相似，它也是从随机解出发，通过迭代寻找最优解，它也是通过适应度来评价解的品质，但它比遗传算法规则更为简单，它没有遗传算法的“交叉”和“变异”操作，它通过追随当前搜索到的最优值来寻找全局最优。

3.1.1 粒子群算法的发展

1995年美国电气工程师Eberhart和社会心理学家Kenndy基于鸟群觅食行为提出了粒子群优化算法(PSO)，简称粒子群算法。由于该算法概念简明、实现方便、收敛速度快、参数设置少，是一种高效的搜索算法。

PSO是模拟鸟群随机搜寻食物的捕食行为。假设在搜索食物区域里只有一块食物，所有的小鸟都不知道食物在什么地方，所以Kenndy等认为鸟之间存在着互相交换信息，通过估计自身的适应度值，它们知道当前的位置离食物还有多远，所以搜索目前离食物最近

的鸟的周围区域是找到食物的最简单有效的办法，通过鸟之间的集体协作使群体达到最优。

PSO就是从这种模型中得到启示并用于解决优化问题。在PSO中每个优化问题的潜在解都可以想象成搜索空间中的一只鸟，称之为“粒子”。粒子主要追随当前的最优粒子在解空间中搜索，PSO初始化为一群随机粒子(随机解)，然后通过迭代找到最优解。

在每一次迭代中，粒子通过跟踪两个“极值”来更新自己，第一个就是粒子本身所找到的最优解，这个解称为个体极值pbest，另一个极值是整个种群目前找到的最优解，这个极值是全局极值gbest。

这两个最优变量使得鸟在某种程度上朝着这些方向靠近，此外也可以不用整个种群而只用其中一部分作为粒子的邻居，那么所有邻居的极值就是局部极值，粒子始终跟随这两个极值变更自己的位置和速度，直到找到最优解。

到目前为止，粒子群算法的发展得到越来越多的众多领域学者的关注和研究，成为解决许多问题的热点算法的研究重点。

其中对PSO算法的改进也非常多，有增强算法自适应性的改进、增强收敛性的改进、增加多种群多样性的改进、增强局部搜索的改进、与全局优化算法相结合、与确定性的局部优化算法相融合等。

以上所述的是对于算法改进的目的的讨论，实际改进中应用的方法有基于参数的改进，即对PSO算法的迭代公式的形式上做改进；还有从粒子的行为模式进行改进，即粒子之间的信息交流方式，如拓扑结构的改进、全局模式与局部模式相结合的改进等；还有基于算法融合的粒子群算法的改进，算法融合可以引入其他算法的优点来弥补PSO算法的缺点，设计出更适合问题求解的优化算法。

目前，粒子群算法的发展趋势如下。

(1) 粒子群优化算法的改进。粒子群优化算法在解决空间函数的优化问题和单目标优化问题上应用得比较多，如何应用于离散空间优化问题和多目标优化问题将是粒子群优化算法的主要研究方向。如何充分结合其他进化类算法，发挥优势，改进粒子群优化算法的不足也是值得研究的。

(2) 粒子群优化算法的理论分析。粒子群优化算法提出的时间不长，数学分析很不成熟和系统，存在许多不完善和未涉及的问题，对算法运行行为、收敛性、计算复杂性的分析比较少。如何知道参数的选择和设计，如何设计适应值函数，如何提高算法在解空间搜索的效率算法收敛以及对算法模型本身的研究都需要在理论上进行更深入的研究。这些都是粒子群优化算法的研究方向之一。

(3) 粒子群算法的生物学基础。如何根据群体进行行为完善算法，将群体智能引入算法中，借鉴生物群体进化规则和进化的智能性也是学者关注的问题。

(4) 粒子群优化算法与其他进化类算法的比较研究。与其他进化算法的融合，如何将其他进化算法的优点和粒子群优化算法相结合，构造出有特色有实用价值的混合算法是当前算法改进的一个重要方向。

(5) 粒子群优化算法的应用。算法的有效性必须在应用中才能体现，广泛地开拓粒子群优化算法的应用领域，也对深入研究粒子群优化算法非常有意义。

3.1.2 粒子群算法研究内容

粒子群算法是一个非常简单的算法，且能够有效地优化各种函数。从某种程度上说，此算法介于遗传算法和进化规划之间。

此算法非常依赖于随机的过程，这也是和进化规划的相识之处，算法中朝全局最优和局部最优靠近的调整非常类似于遗传算法中的交叉算子。

粒子群算法的主要研究内容如下。

(1) 寻找全局最优点。

(2) 有较高的收敛速度。

算法还是用了适应值的概念，这是所有进化计算方法所共有的特征。

3.1.3 粒子群算法的特点

粒子群算法的本质是一种随机搜索算法，它是一种新兴的智能优化技术，是群体智能中一个新的分支，它也是对简单社会系统的模拟。

该算法能以较大的概率收敛于全局最优解。实践证明，它适合在动态、多目标优化环境中寻优，与传统的优化算法相比较具有更快的计算速度和更好的全局搜索能力。其具体特点如下：

(1) 粒子群优化算法是基于群体智能理论的优化算法，通过群体中粒子间的合作与竞争产生的群体智能指导优化搜索。与进化算法比较，PSO 是一种更为高效的并行搜索算法。

(2) PSO 与 GA 有很多共同之处，两者都是随机初始化种群，使用适应值来评价个体的优劣程度和进行一定的随机搜索。但 PSO 是根据自己的速度来决定搜索，没有 GA 的明显交叉和变异。与进化算法比较，PSO 保留了基于种群的全局搜索策略，但是其采用的速度-位移模型操作简单，避免了复杂的遗传操作。

(3) 由于每个粒子在算法结束时仍然保持着其个体极值。因此，若将 PSO 用于调度和决策问题时可以给出多种有意义的选择方案。而基本遗传算法在结束时，只能得到最后一代个体的信息，前面迭代的信息没有保留。

(4) PSO 特有的记忆使其可以动态地跟踪当前的搜索情况并调整其搜索策略。

(5) PSO 有良好的机制来有效地平衡搜索过程的多样性和方向性。

(6) 在收敛的情况下，由于所有的粒子都向最优解的方向飞去，所以粒子趋向同一化(失去了多样性)使得后期收敛速度明显变慢，以致算法收敛到一定精度时无法继续优化。因此很多学者都致力于提高 PSO 算法的性能。

(7) PSO 算法对种群大小不十分敏感，即种群数目下降时性能下降不是很大。

3.1.4 粒子群算法的应用

粒子群算法提供了一种求解复杂系统优化问题的通用框架，它不依赖于问题的具体领域，对问题的种类有很强的适应性，所以广泛应用于很多学科。粒子群算法的一些主

要应用领域如下。

(1) 约束优化。随着问题的增多,约束优化问题的搜索空间也急剧变换,有时在目前的计算机上用枚举法很难或甚至不可能求出其精确最优解。粒子群算法是解决这类问题的最佳工具之一。实践证明,粒子群算法对于约束优化中的规划,离散空间组合问题的求解非常有效。

(2) 函数优化。是粒子群算法的经典应用领域,也是对粒子群算法进行性能评价的常用算例。

(3) 机器人智能控制。机器人是一类复杂的难以精确建模的人工系统,而粒子群算法可用于此类机器人群搜索,如机器人的控制与协调,移动机器人路径规划。所以机器人智能控制理所当然地成为粒子群算法的一个重要应用领域。

(4) 电力系统领域。在其领域中有种类多样的问题,根据目标函数特性和约束类型许多与优化相关的问题需要求解。PSO 在电力系统方面的应用如配电网扩展规划、检修计划、机组组合等。随着粒子群优化理论研究的深入,它还将在电力市场竞价交易等其他领域发挥巨大的应用潜在力。

(5) 工程设计问题。在许多情况下所建立起来的数学模型难以精确求解,即使经过一些简化之后可以进行求解,也会因简化得太多而使得求解结果与实际相差甚远。现在粒子群算法已成为解决复杂调度问题的有效工具,在电路及滤波器设计、神经网络训练、控制器设计与优化、任务分配等方面粒子群算法都得到了有效的应用。

(6) 生物医学领域。许多菌体的生长模型即为非线性模型提出了用粒子群算法解决非线性模型的参数估计问题。还在分子力场的参数设定和蛋白质图形的发现。根据粒子群算法提出的自适应多峰生物测定融合算法,提高了解决问题的准确性。在医学方面,如医学成像上得到的推广应用等。

(7) 通信领域。包括路由选择及移动通信基站布置优化,在顺序码分多址连接方式(DS-CDMA)通信系统中使用粒子群算法,可获得可移植的有力算法并提供并行处理能力。比传统先前的算法有了显著的优越性,还可以应用到天线阵列控制和偏振模色散补偿等方面。

(8) 交通运输领域。在物流配送供应领域中要求以最少的车辆数、最小的车辆总行程来完成货物的派送任务;在交通控制控制领域,城市交通问题是困扰城市发展、制约城市经济建设的重要因素。

3.2 基本粒子群算法

PSO 算法是起源对简单社会系统的模拟,具有很好的生物社会背景而易理解、参数少而易实现,对非线性、多峰问题均具有较强的全局搜索能力,在科学研究与工程实践中得到了广泛关注。同时,PSO 是一种很好的优化工具。

3.2.1 基本原理

PSO 从这种模型中得到启示并用于解决优化问题。PSO 中,每个优化问题的潜在解

都是搜索空间中的一只鸟，称之为粒子。所有的粒子都有一个由被优化的函数决定的适值(fitness value)，每个粒子还有一个速度决定它们“飞行”的方向和距离。然后粒子就追随当前的最优粒子在解空间中搜索。

粒子位置的更新方式如图 3-1 所示。

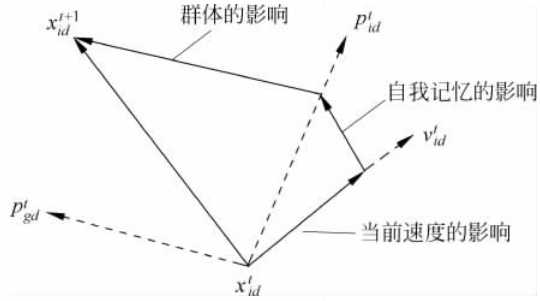

图 3-1　每代粒子位置的更新方式

图 3-1 中，x 表示粒子起始位置；v 表示粒子“飞行”的速度；p 表示搜索到的粒子的最优位置。

PSO 初始化为一群随机粒子(随机解)，然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个极值来更新自己：第一个就是粒子本身所找到的最优解，这个解称为个体极值；另一个极值是整个种群目前找到的最优解，这个极值是全局极值。另外，也可以不用整个种群而只是用其中一部分作为粒子的邻居，那么在所有邻居中的极值就是局部极值。

假设在一个 D 维的目标搜索空间中，有 N 个粒子组成一个群落，其中第 i 个粒子表示为一个 D 维的向量

$$\boldsymbol{X}_i = (x_{i1}, x_{i2}, \cdots, x_{iD}), \quad i = 1, 2, \cdots, N$$

第 i 个粒子的“飞行”速度也是一个 D 维的向量，记为

$$\boldsymbol{V}_i = (v_{i1}, v_{i2}, \cdots, v_{iD}), \quad i = 1, 2, \cdots, N$$

第 i 个粒子迄今为止搜索到的最优位置称为个体极值，记为

$$p_{ibest} = (p_{i1}, p_{i2}, \cdots, p_{iD}), \quad i = 1, 2, \cdots, N$$

整个粒子群迄今为止搜索到的最优位置为全局极值，记为

$$g_{best} = (p_{g1}, p_{g2}, \cdots, p_{gD})$$

在找到这两个最优值时，粒子根据如下的公式来更新自己的速度和位置：

$$v_{id} = w * v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \tag{3-1}$$

$$x_{id} = x_{id} + v_{id} \tag{3-2}$$

其中，c_1 和 c_2 为学习因子，也称加速常数(acceleration constant)，r_1 和 r_2 为[0,1]范围内的均匀随机数。

式(3-1)右边由以下三部分组成。

第一部分为惯性(inertia)或动量(momentum)部分，反映了粒子的运动习惯(habit)，代表粒子有维持自己先前速度的趋势。

第二部分为认知(cognition)部分，反映了粒子对自身历史经验的记忆(memory)或回

忆(remembrance),代表粒子有向自身历史最佳位置逼近的趋势。

第三部分为社会(social)部分,反映了粒子间协同合作与知识共享的群体历史经验,代表粒子有向群体或邻域历史最佳位置逼近的趋势。

由于粒子群算法具有高效的搜索能力,有利于得到多目标意义下的最优解;通过代表整个解集种群,按并行方式同时搜索多个非劣解,也即搜索到多个 Pareto 最优解。

同时,粒子群算法的通用性比较好,适合处理多种类型的目标函数和约束,并且容易与传统的优化方法结合,从而改进自身的局限性,更高效地解决问题。因此,将粒子群算法应用于解决多目标优化问题上具有很大的优势。

3.2.2 算法构成要素

基本粒子群算法构成要素主要包括以下三个方面。

1. 粒子群编码方法

基本粒子群算法使用固定长度的二进制符号串来表示群体中的个体,其等位基因是由二值符号集{0,1}所组成的。初始群体中各个个体的基因值可用均匀分布的随机数来生成。

2. 个体适应度评价

通过确定局部最优迭代达到全局最优收敛,得出结果。

3. 基本粒子群算法的运行参数

基本粒子群算法有下述 7 个运行参数需要提前设定。

(1) r:粒子群算法的种子数,对粒子群算法其中种子数值可以随机生成,也可以固定为一个初始的数值,要求能涵盖目标函数的范围内。

(2) m:粒子群群体大小,即群体中所含个体的数量,一般取 20～100。在变量比较多的时候可以取 100 以上较大的数。

(3) max_d:一般为最大迭代次数以最小误差的要求满足的。粒子群算法的最大迭代次数也是终止条件数。

(4) r_1,r_2:两个在[0,1]之间变化的加速度权重系数随机产生。

(5) c_1,c_2:加速度数,取随机 2 左右的值。

(6) w:惯性权重产生的。

(7) v_k,x_k:一个粒子的速度和位移数值,用粒子群算法迭代出每一组数值。

3.2.3 算法参数设置

PSO 算法一个最大的优点是不需要调节太多的参数,但是算法中少数几个参数却直接影响着算法的性能以及收敛性。目前,PSO 算法的理论研究尚处于初始阶段,所以算法的参数设置在很大程度上还依赖于经验。

PSO参数包括群体规模 m、微粒子长度 l、微粒范围$[-x_{max},x_{max}]$、微粒最大速度 v_{max}、惯性权重 ω、加速常数 c_1 和 c_2。

群体规模 m 一般取20～40。试验表明，对于大多数问题来说，30个微粒就可以取得很好的结果，不过对于比较难的问题或者特殊类别的问题，微粒数目可以取100或200。微粒数目越多，算法搜索的空间范围就越大，也就更容易发现全局最优解。当然，算法运行的时间也越长。

微粒长度 l。即每个微粒的维数，由具体优化问题而定。

微粒范围$[-x_{max},x_{max}]$。微粒范围由具体优化问题决定，通常将问题的参数取值范围设置为微粒的范围。同时，微粒每一维也可以设置不同的范围。

微粒最大速度 v_{max}。微粒最大速度决定了微粒在一次飞行中可以移动的最大距离。如果 v_{max} 太大，微粒可能会飞过好解；如果 v_{max} 太小，微粒不能在局部好区间之外进行足够的搜索，导致陷入局部最优值。通常设定 $v_{dmax}=k\cdot x_{dmax}$，$0.1\leqslant k\leqslant 1$，每一维都采用相同的设置方法。

惯性权重 f_w：f_w 使微粒保持运动惯性，使其有扩展搜索空间的趋势，有能力探索新的区域。取值范围通常为$[0.2,1.2]$。早期的实验将 f_w 固定为1，动态惯性权重因子能够获得比固定值更为优越的寻优结果，使算法在全局搜索前期有较高的探索能力以得到合适的种子。

动态惯性权重因子可以在PSO搜索过程中线性变化，亦可根据PSO性能的某个测度函数而动态改变，比如模糊规则系统。目前采用较多的动态惯性权重因子是Shi建议的线性递减权值策略，即

$$f_w = f_{max} - (f_{max} - f_{min})/\max\_d$$

其中，max_d为最大进化代数；f_{max}为初始惯性值；f_{min}为迭代至最大代数时的惯性权值。经典取值 $f_{max}=0.8$，$f_{min}=0.2$。

加速常数 c_1 和 c_2。c_1 和 c_2 代表将每个微粒的统计加速项的权重。低的值允许微粒在被拉回之前可以在目标区域外徘徊，而高的值则导致微粒突然的冲向或越过目标区域。c_1 和 c_2 是固定常数，早期实验中一般取2。有些文献也采取了其他取值，但一般都限定 c_1 和 c_2 相等并且取值范围为$[0,4]$。

3.2.4 算法的基本流程

基本粒子群算法的流程图如图3-2所示，其具体过程如下。

(1) 初始化粒子群，包括群体规模 N，每个粒子的位置 x_i 和速度 v_i。

(2) 计算每个粒子的适应度值 Fit[i]。

(3) 对每个粒子，用它的适应度值 Fit[i] 和个体极值 $p_{best}(i)$ 比较，如果 Fit[i]$>$ $p_{best}(i)$，则用 Fit[i] 替换掉 $p_{best}(i)$。

(4) 对每个粒子，用它的适应度值 Fit[i] 和全局极值 $g_{best}(i)$ 比较，如果 Fit[i]$>$ $g_{best}(i)$ 则用 Fit[i] 替 $g_{best}(i)$。

(5) 根据式(3-1)和式(3-2)更新粒子的位置 x_i 和速度 v_i。

(6) 如果满足结束条件(误差足够好或到达最大循环次数)退出，否则返回(2)。

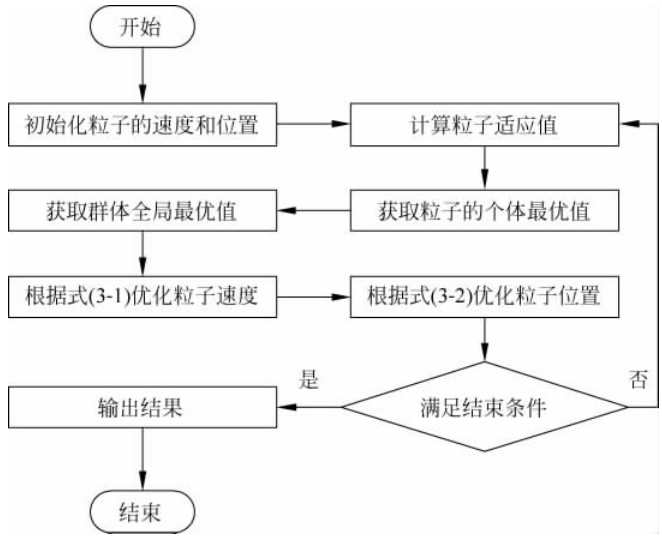

图 3-2　PSO 算法流程图

3.2.5　算法的 MATLAB 实现

在 MATLAB 中编程实现的基本粒子群算法基本函数为 PSO。其调用格式如下：

```
[xm,fv] = PSO(fitness,N,c1,c2,w,M,D)
```

其中，fitness 为待优化的目标函数，也称适应度函数；N 是粒子数目；c1 是学习因子 1；c2 是学习因子 2；w 是惯性权重；M 是最大迭代数；D 是自变量的个数；xm 是目标函数取最小值时的自变量；fv 是目标函数的最小值。

使用 MATLAB 实现基本粒子群算法是代码如下：

```
function[xm,fv] = PSO(fitness,N,c1,c2,w,M,D)
%%%%%%%% 给定初始化条件 %%%%%%%%%%
% c1 学习因子 1
% c2 学习因子 2
% w 惯性权重
% M 最大迭代次数
% D 搜索空间维数
% N 初始化群体个体数目
%%%%% 初始化种群的个体(可以在这里限定位置和速度的范围) %%%%%%%%%%
format long;
for i = 1:N
    for j = 1:D
        x(i,j) = randn;                    % 随机初始化位置
        v(i,j) = randn;                    % 随机初始化速度
    end
end
%%%%%% 先计算各个粒子的适应度,并初始化 Pi 和 Pg %%%%%%%%%%%%%%%%%
```

```
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                          %Pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:)) < fitness(pg)
        pg = x(i,:);
    end
end
%%%%% 进入主要循环,按照公式依次迭代,直到满足精度要求 %%%%%
for t = 1:M
    for i = 1:N                       %更新速度、位移
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:)) < p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)< fitness(pg)
            pg = y(i,:);
        end
    end
    Pbest(t) = fitness(pg);
end
%%%%%% 最后给出计算结果
disp('**************************************************')
disp('目标函数取最小值时的自变量:')
xm = pg'
disp('目标函数的最小值为:')
fv = fitness(pg)
disp('**************************************************')
```

将上面的函数保存到 MATLAB 可搜索路径中,即可调用该函数。再定义不同的目标函数 fitness 和其他输入量,就可以用粒子群算法求解不同问题。

粒子群算法使用的函数有很多种,下面介绍两个常用的适应度函数。

1. Griewank 函数

该函数的 MATLAB 代码如下:

```
function y = Griewank(x)
%Griewan 函数
%输入 x,给出相应的 y 值,在 x = (0,0,…,0)处有全局极小点 0
[row,col] = size(x);
if row > 1
    error('输入的参数错误');
end
y1 = 1/4000 * sum(x.^2);
```

```
y2 = 1;
for h = 1:col
    y2 = y2 * cos(x(h)/sqrt(h));
end
y = y1 - y2 + 1;
y = - y;
```

绘制以上函数图形的 MATLAB 代码如下：

```
function DrawGriewank()
% 绘制 Griewank 函数图形
x = [ - 8:0.1:8];
y = x;
[X,Y] = meshgrid(x,y);
[row,col] = size(X);
for l = 1:col
   for h = 1:row
       z(h,l) = Griewank([X(h,l),Y(h,l)]);
   end
end
surf(X,Y,z);
shading interp
```

将以上代码保存为 DrawGriewank.m 文件，并运行上述代码，得到 Griewank 函数图像如图 3-3 所示。

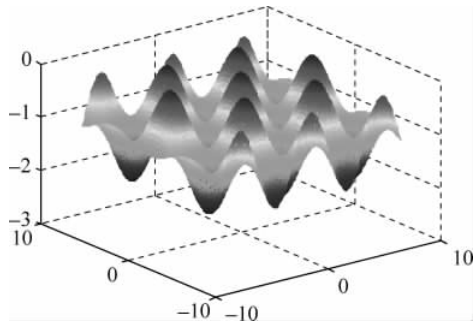

图 3-3　Griewank 函数图像

2. Rastrigin 函数

该函数 MATLAB 代码如下：

```
function y = Rastrigin(x)
% Rastrigin 函数
% 输入 x,给出相应的 y 值,在 x = (0,0,…,0)处有全局极小点 0
[row,col] = size(x);
if row > 1
    error('输入的参数错误');
```

```
end
y = sum(x.^2 - 10 * cos(2 * pi * x) + 10);
y = - y;
```

绘制以上函数图的函数 MATLAB 代码如下：

```
function DrawRastrigin()
x = [ - 4:0.05:4];
y = x;
[X,Y] = meshgrid(x,y);
[row,col] = size(X);
for l = 1:col
    for h = 1:row
        z(h,l) = Rastrigin([X(h,l),Y(h,l)]);
    end
end
surf(X,Y,z);
shading interp
```

运行上述代码，得到 Rastrigin 函数图像如图 3-4 所示。

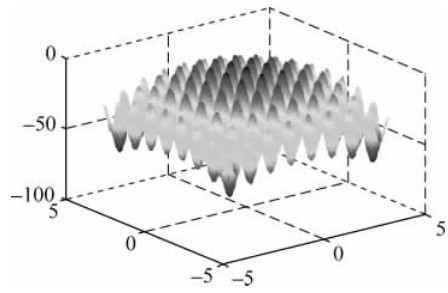

图 3-4　Rastrigin 函数图像

【例 3-1】 利用上面介绍的基本粒子群算法求解下列函数的最小值：

$$f(x) = \sum_{i=1}^{30} x_i^2 + x_i - 6$$

解　利用 PSO 算法求解最小值，需要首先确认不同迭代步数对结果的影响。设定题中函数的最小点均为 0，粒子群规模为 50，惯性权值为 0.5，学习因子 1 为 1.5，学习因子 2 为 2.5，迭代步数分别取为 100、1000、10000。

在 MATLAB 中建立目标函数代码如下：

```
function F = fitness(x)
F = 0;
for i = 1:30
    F = F + x(i)^2 + x(i) - 6
end
```

在 MATLAB 命令行窗口输入如下代码：

```
>> x = zeros(1,30);
>> [xm1,fv1] = PSO(@fitness,50,1.5,2.5,0.5,100,30);
>> [xm2,fv2] = PSO(@fitness,50,1.5,2.5,0.5,1000,30);
>> [xm3,fv3] = PSO(@fitness,50,1.5,2.5,0.5,10000,30);
```

运行以上代码，比较目标函数取最小值时的自变量，如表 3-1 所示。

表 3-1 比较不同迭代步数下的目标函数值和最小值

迭代步数	100	1000	10000
x_1	−0.203470374827947	−0.322086628880754	−0.435569044575185
x_2	0.0614316795653017	−0.236499213027137	−0.456597706978179
x_3	−0.432057786059138	−0.174672457595542	−0.272548635326235
x_4	−0.562337192754589	−0.323434573711674	−0.410028636513352
x_5	0.216285572045985	−0.559755785428548	−0.478395017745105
x_6	−0.448174496712675	−0.500724696101979	−0.438617720718304
x_7	0.0101008034620691	−0.334601378057723	−0.624351586431356
x_8	−0.359780035841033	−0.599261558115410	−0.542835397138839
x_9	−0.244678550463580	−0.689138008554286	−0.243113131019114
x_{10}	−0.316139905200595	−0.0694954358421096	−0.143940233374031
x_{11}	−0.408639179789461	−0.259841700046576	−0.706252186322252
x_{12}	−0.642619836410718	−0.246141170661282	−0.00781653355911016
x_{13}	−0.522925465434690	−0.449585957090094	−0.334838983888102
x_{14}	−0.203441587074036	−0.406235920268046	−0.104353647362726
x_{15}	−0.563308887343590	0.0891778287549033	−0.438931696076205
x_{16}	−0.301808274435673	−0.0303852886125965	−0.177440809228911
x_{17}	−0.709768167671245	−0.552156841443132	−0.621428723324555
x_{18}	−0.420233565717631	−0.354652539291389	−0.321409146325643
x_{19}	−0.0649786155553592	−0.473586592491481	−0.340215630334193
x_{20}	0.0835405545618331	−0.542947832512436	−0.435868961230739
x_{21}	−0.677113366792996	−0.571165888709759	−0.402359314141048
x_{22}	−0.288800585542166	−0.235227313009656	−0.663112621839921
x_{23}	−0.423115455971755	−0.783184021012424	−0.243847375888005
x_{24}	−0.483611573904200	−0.610977611626016	−0.372767988055409
x_{25}	−0.296101193584627	−0.0762667490397894	−0.588328193723098
x_{26}	−0.364523672340500	−0.389593896038030	−0.310699752647837
x_{27}	−0.217234643531979	−0.152204938081090	−0.474445660596261
x_{28}	0.0562371091188502	−0.812638082215613	−0.0836301944341218
x_{29}	−0.507805752603469	−0.661787823700067	−0.0284228008093966
x_{30}	−0.0208750670471909	−0.145197593442009	−0.397530666505423
目标函数最小值 fv	−184.692117109108	−184.692117109108	−184.692117109108

从表 3-1 可以看出，迭代步数不一定与获得解的精度成正比，即迭代步数越大，获得解的精度不一定越高。这是因为 PSO 算法是一种随机算法，同样的参数也会算出不同的结果。

在上述参数基础上，保持惯性权重 0.5、学习因子 1 为 1.5、学习因子 2 为 2.5、迭代步数为 100 不变，粒子群规模分别取 10、100 和 500，运行以下 MATLAB 代码：

```
>> x = zeros(1,30);
>> [xm1,fv1] = PSO(@fitness,10,1.5,2.5,0.5,100,30);
>> [xm2,fv2] = PSO(@fitness,100,1.5,2.5,0.5,100,30);
>> [xm3,fv3] = PSO(@fitness,500,1.5,2.5,0.5,100,30);
```

比较目标函数取最小值时的自变量，如表 3-2 所示。

表 3-2　比较不同粒子群规模下的目标函数值和最小值

粒子群规模	10	100	500
x_1	−0.461280538391346	−0.568652265006235	−0.490268156078420
x_2	−0.408370995746921	−0.452788770991822	−0.495317061863384
x_3	−0.0288416005345963	−0.388174768325847	−0.508017090877808
x_4	−0.0552338567227231	−0.401507545198533	−0.517007413849568
x_5	0.0738166644789645	−0.551259879300365	−0.477354073247202
x_6	−0.280868118500682	−0.233393064263199	−0.496014962954584
x_7	−0.429600925039530	−0.271896675443476	−0.489607302876620
x_8	−0.409562596099239	−0.547844449351226	−0.493034422510953
x_9	0.281766017074388	−0.380278337003657	−0.491570275741791
x_{10}	−0.587883598964542	−0.408568766862200	−0.505298045536549
x_{11}	−0.749463199823461	−0.626782730867803	−0.503117287033364
x_{12}	0.0779478416748528	−0.349408282182953	−0.494031258256908
x_{13}	−0.758300631146907	−0.583408316780879	−0.500060685658700
x_{14}	0.180131709578965	−0.375383139040645	−0.511709156436812
x_{15}	−0.564532674933458	−0.490162739466452	−0.517812810910794
x_{16}	−0.0637266236855537	−0.555105474483478	−0.504355035662881
x_{17}	0.501801473477060	−0.560793363305467	−0.511495990026503
x_{18}	0.583049171640106	−0.641197096800355	−0.519087838941761
x_{19}	0.423066993306820	−0.594790333100089	−0.497402575677108
x_{20}	0.463031353118403	−0.517368663564588	−0.506039272612501
x_{21}	−0.226652573205321	−0.647922715489912	−0.493311227454402
x_{22}	−0.340694973324611	−0.493043901761973	−0.492860555794895
x_{23}	0.303590596927068	−0.445059333754872	−0.499654192041048
x_{24}	−0.0372694887364219	−0.602557014069339	−0.494888427804042
x_{25}	−0.119240515687260	−0.439982689177553	−0.519431562496152
x_{26}	0.511293600728549	−0.260811072394469	−0.493925264779633
x_{27}	0.115534647931772	−0.738686510406502	−0.488810925337222
x_{28}	0.559536823964912	−0.494057140638969	−0.489181575636495
x_{29}	0.446461621552828	−0.378395529426522	−0.498224198470959
x_{30}	−0.359535394729040	−0.402673857684666	−0.514332244824747
目标函数最小值 fv	−176.440172293181	−187.045295621546	−187.496699775657

从表 3-2 中可以看出，粒子群规模越大，获得解的精度不一定越高。

综合以上不同迭代步数和不同粒子群规模运算得到的结果可知，在粒子群算法中，要想获得精度高的解，关键在于各个参数之间的合适搭配。

3.3　MATLAB 粒子群工具箱

PSO 具有很强的灵活性，在实际计算中，用户只需要编写目标函数和设置参数范围，PSO 就可以自动进行优化计算。

PSOt 为 PSO 的工具箱，它将 PSO 算法的核心部分封装起来后，提供算法的可调参数给用户选择。在定义好需要优化的函数（计算最小值或者最大值），并设置好函数自变量的取值范围、每步迭代允许的最大变化量（称为最大速度，Max_V）等参数后，算法就会自行优化。

因为 PSO 算法对待优化的函数没有过多要求（如可微分、时间连续等），且仅需要调整少数几个参数即可实现函数的优化，所以其通用性极强，对多变量、高度非线性、不连续及不可微的情况更加具有其优势。

PSO 工具箱的使用主要分为以下三个步骤。

1. 在 MATLAB 中设置工具箱的路径

在 MATLAB 上方单击 Set Path 按键，弹出如图 3-5 所示的对话框。

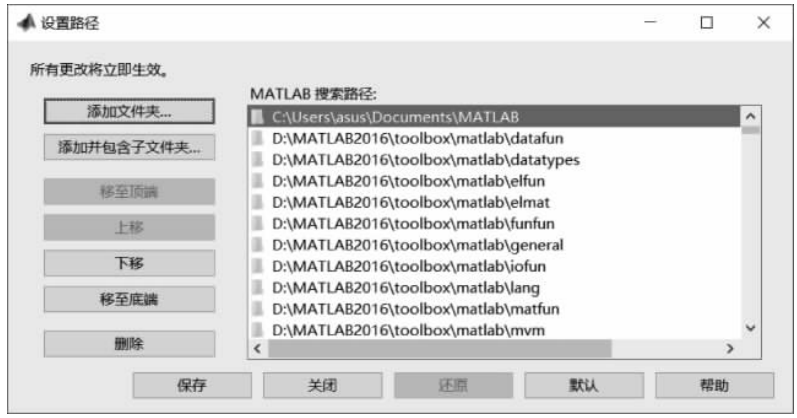

图 3-5　设置路径对话框

在图 3-5 中选择“添加文件夹”选项，选择粒子群算法 PSOt 工具箱所在的位置，如图 3-6 所示。单击“选择文件夹”按钮后即可将工具箱添加到 MATLAB 路径中。

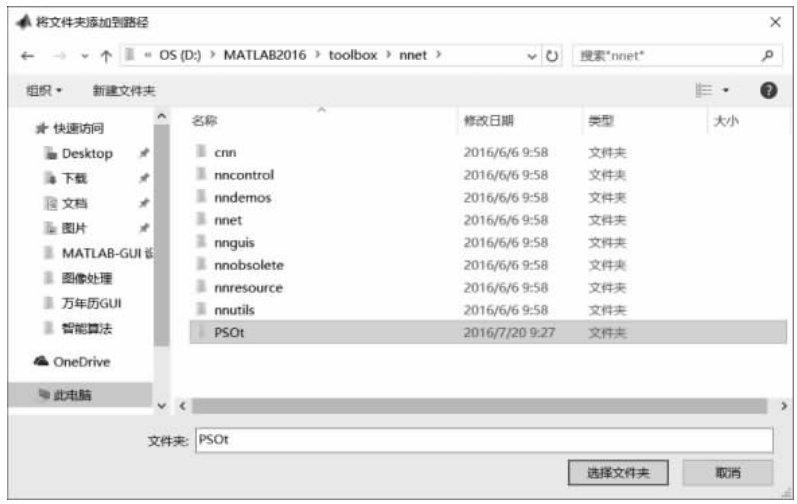

图 3-6　选择粒子群算法 PSOt 工具箱路径

注意：在设置路径的时候，需要把 PSOt 工具箱文件夹里面其他的文件夹也添加到 MATLAB 路径中，否则会导致后面程序运行的报错。

2. 定义待优化函数

在 MATLAB 粒子群工具箱调用时，用户可以根据自己的需要，定义需要优化的函数。例如，计算如下函数的最小值：

$z = 0.4 * (x-2)^2 + 0.3 * (y-4)^2 - 0.7, \quad x \in [-40,40], y \in [-40,40]$

定义待优化函数的 MATLAB 代码如下：

```
function z = pso_func(in)
n = size(in);
x = in(:,1);
y = in(:,2);
nx = n(1);
for i = 1:nx
   temp = 0.4 * (x(i) - 2)^2 + 0.3 * (y(i) - 4)^2 - 0.7;
   z(i,:) = temp;
end
```

3. 调用 PSO 算法的核心函数：pso_Trelea_vectorized

当定义好待优化函数并设置好相应的参数后，就可以调用 PSO 函数进行优化，对上面优化问题，按下面的方式进行调用：

```
clear
clc
x_range = [ - 40,40];                                  % 参数 x 变化范围
y_range = [ - 40,40];                                  % 参数 y 变化范围
range = [x_range;y_range];                             % 参数变化范围
Max_V = 0.2 * (range(:,2) - range(:,1));               % 最大速度取变化范围的 10 % ～20 %
n = 2;                                                 % 待优化函数的维数
pso_Trelea_vectorized('pso_func',n,Max_V,range)        % 调用 PSO 核心函数
```

注意：在以上三步中，第三步最关键，其需要根据自己的需要设置好参数，可使算法极快收敛。

在 PSO 算法函数 pso_Trelea_vectorized 中，PSO 参数设置如下：

```
Pdef = [100 2000 24 2 2 0.9 0.4 1500 1e - 25 250 NaN 0 1];
```

其中，100 表示在 MATLAB 命令窗进行显示的间隔数，即每迭代 100 次显示一次，若取值为 0，则不显示中间过程。

2000 表示最大迭代次数，即使算法不收敛，到此数后自动停止。

24 表示种子数，即初始化多少个种子。种子数越多，越有可能收敛到全局最优值，但算法收敛速度慢。

2 表示算法的加速度参数，分别影响局部最优值和全局最优值，一般不需要修改。

0.9 和 0.4 为初始时刻和收敛时刻的加权值，在最早的 PSO 算法中，没有此参数，靠其他几个参数的调整来保证收敛，但收敛速度和收敛精度难以同时满足，后来在改进算法中，加入此权值，使得兼顾收敛速度和收敛精度成为可能，一般不需要修改。

1500 表示当迭代次数超过此值时，加权值取其最小。

1e－25 为算法终止条件之一。当连续的两次迭代中对应的种群最优值小于此阈值时，算法停止。

250 为算法终止条件之一。当连续 250 次迭代中函数的梯度值仍然没有变化，则退出迭代。

NaN 用于说明优化的情况，取 NaN 时表示为非约束下的优化问题（即没有附加约束方程）。

0 是指定采用何种 PSO 类型，0 表示通常的 PSO 算法。

1 是说明是否认指定种子，0 表示随机产生种子，1 表示用户自行产生种子。

完成以上三步之后，可以直接执行 pso_main.m，得到结果如下：

```
ans =
    2.999999999719954
    5.000000002343016
   -0.100000000000000
```

其中，前面两个值对应于收敛时的自变量 x 和 y 的取值。最后一个值为收敛时对应的 z 的最优值。

3.4 权重改进的粒子群算法

在粒子群算法中，惯性权重 w 是最重要的参数。增大 w 的值可以提高算法的全局搜索能力，减小 w 的值可以提高算法的局部搜索能力。设计合理的惯性权重，是避免陷入局部最优并高效搜索的关键。

常见的 POS 算法有自适应权重法、随机权重法和线性递减权重法等。

3.4.1 自适应权重法

粒子适应度是反应粒子当前位置优劣的一个参数。对应某些具有较高适应度的粒子 p_i，在 p_i 所在的局部区域可能存在能够更新全局最优的点 p_x，即 p_x 表示的解要优于全局最优。

为了使全局最优能够迅速更新，从而迅速找到 p_x，应该减小粒子 p_i 惯性权重，以增强其局部寻优能力；而对于适应度较低的粒子，当前位置较差，所在区域存在优于全局最优解的概率较低，为了跳出当前的区域，应当增大惯性权重，增强全局搜索能力。

下面介绍自适应修改权重的两种方法。

1. 依据早熟收敛程度和适应值进行调整

根据群里的早熟收敛程度和个体适应值，可以确定惯性权重的变化。

设定粒子 p_i 的适应值为 f_i，最优粒子适应度是 f_m，则粒子群的平均适应值是 $f_{avg}=\frac{1}{n}\sum_{i=1}^{n}f_i$；将优于平均适应值的粒子适应值求平均(记为 f'_{avg})，定义 $\Delta=|f_m-f'_{avg}|$。

依据 f_i、f_m、f_{avg}将群体分为 3 个子群，分别进行不同的自适应操作，其惯性权重的调整如下：

(1) 如果 f_i 优于 f'_{avg}，那么

$$w=w-(w-w_{min})\cdot\left|\frac{f_i-f'_{avg}}{f_m-f'_{avg}}\right|$$

(2) 如果 f_i 优于 f'_{avg}，且次于 f_m，则惯性权重不变。

(3) 如果 f_i 次于 f'_{avg}，则

$$w=1.5-\frac{1}{1+k_1\cdot\exp(-k_2\cdot\Delta)}$$

其中，k_1、k_2 为控制参数，k_1 用来控制 w 的上限，k_2 主要用来控制

$$w=1.5-\frac{1}{1+k_1\cdot\exp(-k_2\cdot\Delta)}$$

的调节能力。

当算法停止时，如果粒子的分布分散，则 Δ 比较大，w 变小，此时算法局部搜索能力加强，从而使得群体趋于收敛；若粒子的分布聚集，则 Δ 比较小，w 变大，使得粒子具有较强的探查能力，从而有效地跳出局部最优。

2. 根据全局最优点的距离进行调整

一些学者认为惯性权重的大小还和其距全局最优点的距离有关，并提出了各个不同粒子惯性权重不仅随迭代次数的增加而递减，还随距全局最优点距离的增加而递增，即权重 w 根据粒子的位置不同而动态变化。目前大多采用的非线性动态惯性权重系数公式为

$$w=\begin{cases}w_{min}-\dfrac{(w_{max}-w_{min})*(f-f_{min})}{f_{avg}-f_{min}}, & f\leqslant f_{avg}\\ w_{max}, & f>f_{avg}\end{cases}$$

其中，f 表示粒子实时的目标函数值；f_{avg}和 f_{min}分别表示当前所有粒子的平均值和最小目标值。从上面公式可以看出，惯性权重随着粒子目标函数值的改变而改变。

当粒子目标值分散时，减小惯性权重；粒子目标值一致时，增加惯性权重。

根据全局最优点的距离调整算法的基本步骤如下：

(1) 随机初始化种群中各个粒子的位置和速度。

(2) 评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best}中，将所有 p_{best}中最优适应值的个体位置和适应值保存在全局极值 g_{best}中。

(3) 更新粒子位移和速度：

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,2,\cdots,d$$

$$v_{i,j}(t+1)=w\cdot v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]$$

(4) 更新权重：

$$w=\begin{cases} w_{\min}-\dfrac{(w_{\max}-w_{\min})*(f-f_{\min})}{f_{\text{avg}}-f_{\min}}, & f\leqslant f_{\text{avg}} \\ w_{\max}, & f>f_{\text{avg}} \end{cases}$$

(5) 将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最好的位置。比较当前所有的 p_{best} 和 g_{best}，更新 g_{best}。

(6) 当算法达到其停止条件，则停止搜索并输出结果；否则返回到第(3)步继续搜索。

将实现自适应权重的优化函数命名为 PSO_adaptation，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_adaptation(fitness,N,c1,c2,wmax,wmin,M,D)
format long;
%fitness 学习函数
%c1 学习因子 1
%c2 学习因子 2
%wmax 惯性权重最大值
%wmin 惯性权重最小值
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值

%%%%%%%%%%%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;
        v(i,j) = randn;
    end
end

%%%%%%%%%%%%%%%%先计算各个粒子的适应度%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                    %Pg 表示全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end

%%%%%%%%%%%%%%%%%%进入主要循环%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for t = 1:M
    for j = 1:N
```

```
            fv(j) = fitness(x(j,:));
        end
        fvag = sum(fv)/N;
        fmin = min(fv);
        for i = 1:N
            if fv(i) <= fvag
                w = wmin + (fv(i) - fmin) * (wmax - wmin)/(fvag - fmin);
            else
                w = wmax;
            end
            v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
            x(i,:) = x(i,:) + v(i,:);
            if fitness(x(i,:))< p(i)
                p(i) = fitness(x(i,:));
                y(i,:) = x(i,:);
            end
            if p(i)< fitness(pg)
                pg = y(i,:);
            end
        end
    end
    xm = pg';                % 目标函数取最小值时的自变量
    fv = fitness(pg);        % 目标函数最小值
```

【例 3-2】 用自适应权重法求解函数

$$f(x)=\frac{\sin^2\sqrt{x_1^2+x_2^2}-\cos\sqrt{x_1^2+x_2^2}+1}{[1+0.1(x_1^2+x_2^2)]^2}-0.7$$

的最小值。其中，粒子数为 50，学习因子均为 2，惯性权重取值[0.6　0.8]，迭代步数为 100。

解　首先建立目标函数代码如下：

```
function y = AdaptFunc(x)
   y = ((sin(x(1)^2 + x(2)^2))^2 - cos(x(1)^2 + x(2)^2) + 1)/((1 + 0.1 * (x(1)^2 + x(2)^2))^
2) - 0.7;
End
```

在 MATLAB 命令行窗口输入代码：

```
[xm,fv] = PSO_adaptation(@AdaptFunc,50,2,2,0.8,0.6,100,2)
```

运行后得到结果为：

```
>> xm =
   1.0e - 04 *
  - 0.228381952547433
  - 0.122850392140981
fv =
  - 0.700000000000000
```

3.4.2 随机权重法

随机权重法的原理是将标准 PSO 算法中的惯性权重 w 设定为随机数，这种处理的优势如下。

(1) 当粒子在起始阶段就接近最好点，随机产生的 w 可能为相对较小的值，由此可以加快算法的收敛速度。

(2) 克服 w 线性递减造成的算法不能收敛到最好点的局限。

惯性权重的修改公式为

$$\begin{cases} w = \mu + \sigma * N(0,1) \\ \mu = \mu_{\min} + (\mu_{\max} - \mu_{\min}) * \text{rand}(0,1) \end{cases}$$

其中，$N(0,1)$表示标准状态分布的随机数。

随机权重法的计算步骤如下。

(1) 随机设置各个粒子的速度和位置。

(2) 评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中，将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 更新粒子位移和速度：

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1,2,\cdots,d$$

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 r_1 [p_{i,j} - x_{i,j}(t)] + c_2 r_2 [p_{g,j} - x_{i,j}(t)]$$

(4) 更新权重：

$$\begin{cases} w = \mu + \sigma * N(0,1) \\ \mu = \mu_{\min} + (\mu_{\max} - \mu_{\min}) * \text{rand}(0,1) \end{cases}$$

(5) 将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最好的位置。比较当前所有的 p_{best} 和 g_{best}，更新 g_{best}。

(6) 当算法达到其停止条件，则停止搜索并输出结果，否则返回到第(3)步继续搜索。

将实现自适应权重的优化函数命名为 PSO_rand，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_rand(fitness,N,c1,c2,wmax,wmin,rande,M,D)
format long;
%fitness 学习函数
%c1 学习因子 1
%c2 学习因子 2
%wmax 惯性权重最大值
%wmin 惯性权重最小值
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值
```

```
%rande 随机权重方差

%%%%%%%% 初始化种群的个体 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;
        v(i,j) = randn;
    end
end

%%%%%%%% 先计算各个粒子的适应度,并初始化 Pi 和 Pg %%%%%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                    % pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end

%%%%%%%% 进入主要循环,按照公式依次迭代 %%%%%%%%%%%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        miu = wmin + (wmax - wmin) * rand();
        w = miu + rande * randn();
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))<p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
    end
    Pbest(t) = fitness(pg);
end
xm = pg';
fv = fitness(pg);
```

【例 3-3】 用随机权重法求解例 3-2 中的函数的最小值。其中,粒子数为 50,学习因子均为 2,惯性权重取值[0.6　0.8],随机权重平均值的方差为 0.3,迭代步数为 100。

解　在 MATLAB 命令行窗口输入如下代码:

```
[xm,fv] = PSO_rand(@AdaptFunc,50,2,2,0.8,0.6,0.3,100,2)
```

运行后得到结果如下:

```
xm =
   1.0e-04 *
  -0.304996089542041
  -0.509841630518761
fv =
  -0.700000000000000
```

3.4.3 线性递减权重法

针对PSO算法容易早熟及后期容易在全局最优解附近产生振荡现象，提出了线性递减权重法，即使惯性权重依照线性从大到小地递减，其变化公式为

$$w = w_{\max} - \frac{t * (w_{\max} - w_{\min})}{t_{\max}}$$

其中，$w_{\max}$表示惯性权重最大值；$w_{\min}$表示惯性权重最小值；t表示当前迭代步数。

随机权重法的计算步骤如下。

(1) 随机设置各个粒子的速度和位置。

(2) 评价评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值p_{best}中，将所有p_{best}中最优适应值的个体位置和适应值保存在全局极值g_{best}中。

(3) 更新粒子位移和速度：

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1,2,\cdots,d$$

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 r_1[p_{i,j} - x_{i,j}(t)] + c_2 r_2[p_{g,j} - x_{i,j}(t)]$$

(4) 更新权重：

$$w = w_{\max} - \frac{t * (w_{\max} - w_{\min})}{t_{\max}}$$

(5) 将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最好的位置。比较当前所有的p_{best}和g_{best}，更新g_{best}。

(6) 当算法达到其停止条件，则停止搜索并输出结果，否则返回到第(3)步继续搜索。

将实现自适应权重的优化函数命名为PSO_lin，在MATLAB中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_lin(fitness,N,c1,c2,wmax,wmin,M,D)
format long;
%fitness 学习函数
%c1 学习因子 1
%c2 学习因子 2
%wmax 惯性权重最大值
%wmin 惯性权重最小值
%M 最大迭代次数
%D 搜索空间维数
%N 初始化群体个体数目
%xm 目标函数取最小值时的自变量
```

```
%fv 目标函数最小值

%%%%%%%%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;
        v(i,j) = randn;
    end
end

%%%%%%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                    %pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end

%%%%%%%%%%%%%%%%主循环,按照公式依次迭代%%%%%%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        w = wmax-(t-1)*(wmax-wmin)/(M-1);
        v(i,:) = w*v(i,:) + c1*rand*(y(i,:)-x(i,:)) + c2*rand*(pg-x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))<p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
    end
    Pbest(t) = fitness(pg);
end
xm = pg';
fv = fitness(pg);
```

【例 3-4】 用随机权重法求解函数

$$f(x) = 20\,(x_1^2 - x_2)^2 - (1 - x_2)^2 - 3\,(1 + x_2)^2 + 0.3$$

的函数最小值。其中,粒子数为 50,学习因子均为 2,惯性权重取[0.4　0.8],迭代步数为 1000。

解　首先建立目标函数代码如下:

```
function y = LinFunc(x)
   y = 20*((x(1)^2-x(2))^2)-(1-x(2))^2-3*(1+x(2))^2+0.3;
End
```

在 MATLAB 命令行窗口输入代码：

```
[xm,fv] = PSO_lin(@LinFunc,50,2,2,0.8,0.4,1000,2)
```

运行后得到结果如下：

```
>> xm =
     1
     1
fv =
     0
```

以上结果说明，用线性递减权重的方法得到了精确的最优点。需要注意的是，惯性权重并不是对所有的问题都有效，具体问题需要具体分析最合适的方法。

例如使用线性递减权重法求例 3-2 中的函数，在 MATLAB 命令行窗口中输入以下代码：

```
>> [xm,fv] = PSO_lin(@AdaptFunc,50,2,2,0.8,0.4,1000,2)
```

得到结果如下：

```
>> xm =
   1.0e+03 *
   6.312824580854360
   4.294354896379293
fv =
  -0.700000000000000
```

这个结果显然要比例 3-2 中得到的结果差。

3.5 混合粒子群算法

混合策略就是将其他进化算法、传统优化算法或其他技术应用到 PSO 中，用于提高粒子多样性、增强粒子的全局探索能力，或者提高局部开发能力，增强收敛速度与精度。

常用的粒子群混合方法主要有以下两种。

(1) 利用其他优化技术自适应调整收缩因子/惯性权值、加速常数等。

(2) 将 PSO 与其他进化算法操作算子或其他技术结合。

下面将主要介绍基于杂交、自然选择、免疫的粒子群和模拟退火共 4 种混合粒子群算法。

3.5.1 基于杂交的算法

该算法是借鉴遗传算法中杂交的概念，在每次迭代中，根据杂交率选取指定数量的粒子放入杂交池内，池内的粒子随机地两两杂交，产生同样数目的子代粒子(n)，并用子

代粒子代替父代粒子(m)。子代位置由父代位置进行交叉得到

$$\mathrm{nx} = i * \mathrm{mx}(1) + (1-i) * \mathrm{mx}(2)$$

其中,mx 表示父代粒子的位置;nx 表示子代粒子的位置;i 是 0 到 1 之间的随机数。

子代的速度公式为

$$\mathrm{nv} = \frac{\mathrm{mv}(1)+\mathrm{mv}(2)}{|\mathrm{mv}(1)+\mathrm{mv}(2)|}|\mathrm{mv}|$$

其中,mv 表示父代粒子的速度;nv 表示子代粒子的速度。

基于杂交的混合粒子群算法步骤如下。

(1) 随机设置各个粒子的速度和位置。

(2) 评价评价每个粒子的适应度,将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中,将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 更新粒子位移和速度:

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1,2,\cdots,d$$

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 r_1[p_{i,j} - x_{i,j}(t)] + c_2 r_2[p_{g,j} - x_{i,j}(t)]$$

(4) 将每个粒子的适应值与粒子的最好位置比较,如果相近,则将当前值作为粒子最好的位置。比较当前所有的 p_{best} 和 g_{best},更新 g_{best}。

(5) 根据杂交概率选取指定数量的粒子,并将其放入杂交池中,池中的粒子随机两两杂交产生同样数目的子代粒子,子代的位置和速度的计算公式为

$$\begin{cases} \mathrm{nx} = i * \mathrm{mx}(1) + (1-i) * \mathrm{mx}(2) \\ \mathrm{nv} = \dfrac{\mathrm{mv}(1)+\mathrm{mv}(2)}{|\mathrm{mv}(1)+\mathrm{mv}(2)|}|\mathrm{mv}| \end{cases}$$

其中,保持 p_{best} 和 g_{best} 不变。

(6) 当算法达到其停止条件,则停止搜索并输出结果;否则返回到第(3)步继续搜索。

将实现自适应权重的优化函数命名为 PSO_breed,在 MATLAB 中编写实现以上步骤的代码如下:

```
function [xm,fv] = PSO_breed(fitness,N,c1,c2,w,bc,bs,M,D)
format long;
%fitness 学习函数
%N 初始化群体个体数目
%c1 学习因子 1
%c2 学习因子 2
%w 惯性权重
%bc 杂交概率
%bs 杂交池的大小比例
%M 最大迭代次数
%D 搜索空间维数
%xm 目标函数取最小值时的自变量
%fv 目标函数最小值

%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
```

```
    for j = 1:D
        x(i,j) = randn;              %随机初始化位置
        v(i,j) = randn;              %随机初始化速度
    end
end

%%%%%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                         %pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end

%%%%%%%%%%进入主要循环,按照公式依次迭代%%%%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        v(i,:) = w*v(i,:) + c1*rand*(y(i,:) - x(i,:)) + c2*rand*(pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if fitness(x(i,:))<p(i)
            p(i) = fitness(x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
        r1 = rand();
        if r1 < bc
            numPool = round(bs*N);
            PoolX = x(1:numPool,:);
            PoolVX = v(1:numPool,:);
            for i = 1:numPool
                seed1 = floor(rand()*(numPool-1)) + 1;
                seed2 = floor(rand()*(numPool-1)) + 1;
                pb = rand();
                childx1(i,:) = pb*PoolX(seed1,:) + (1-pb)*PoolX(seed2,:);
                    childv1(i,:) = (PoolVX(seed1,:) + PoolVX(seed2,:))*norm(PoolVX
(seed1,:))/…
                     norm(PoolVX(seed1,:) + PoolVX(seed2,:));
            end
            x(1:numPool,:) = childx1;
            v(1:numPool,:) = childv1;
        end
    end
end
xm = pg';
fv = fitness(pg);
```

【例 3-5】 使用基于杂交的混合粒子群算法，求解函数

$$f(x)=1/\left[1+\sum_{i=1}^{5}\frac{i}{1+(x_i+1)^2}\right]+0.5$$

的最小值，其中$-10\leqslant x_i\leqslant 10$。粒子数为 50，学习因子均为 2，惯性权重为 0.8，杂交概率为 0.8，杂交池比例为 0.1，迭代步数为 1000。

解 建立如下目标函数代码：

```
function y = BreedFunc(x)
y = 0;
for i = 1:5
    y = y + i/(1 + (x(i) + 1)^2)
end
y = 1/(1 + y) + 0.5
```

在 MATLAB 命令行窗口输入如下代码：

```
[xm,fv] = PSO_breed(@BreedFunc,50,2,2,0.8,0.8,0.1,100,5)
```

运行后得到结果为：

```
>> xm =
   -0.999726399607037
   -0.997799921797732
   -0.998059994563479
   -0.998721594444641
   -1.002179739156178
fv =
    0.562500200546709
```

以上结果表明，基于杂交的混合粒子群算法精度也比较高。

3.5.2 基于自然选择的算法

基于自然选择的算法是借鉴自然选择的机理，在每次迭代中，根据粒子群适应值将粒子群排序，用群体中最好的一半粒子替换最差的一半粒子，同时保留原来每个个体所记忆的历史最优值。

基于杂交的混合粒子群算法步骤如下。

(1) 随机设置各个粒子的速度和位置。

(2) 评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中，将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 更新粒子位移和速度：

$$x_{i,j}(t+1)=x_{i,j}(t)+v_{i,j}(t+1),\quad j=1,2,\cdots,d$$

$$v_{i,j}(t+1)=w\cdot v_{i,j}(t)+c_1r_1[p_{i,j}-x_{i,j}(t)]+c_2r_2[p_{g,j}-x_{i,j}(t)]$$

(4) 将每个粒子的适应值与粒子的最好位置比较，如果相近，则将当前值作为粒子最

好的位置。比较当前所有的 p_{best} 和 g_{best}，更新 g_{best}。

(5) 根据适应值对粒子群排序，用群体中最好的一半粒子替换最差的一半粒子，同时保留原来每个个体所记忆的历史最优值。

(6) 当算法达到其停止条件，则停止搜索并输出结果；否则返回到第(3)步继续搜索。

将实现自适应权重的优化函数命名为 PSO_ nature，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_nature(fitness,N,c1,c2,w,M,D)
format long;
%N初始化群体个体数目
%c1 学习因子 1
%c2 学习因子 2
%w 惯性权重
%M 最大迭代次数
%D 搜索空间维数(未知数个数)
%%%%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;                    %初始化位置
        v(i,j) = randn;                    %初始化速度
    end
end
%%%%%%%%%%%%%先计算各个粒子的适应度,并初始化 Pi 和 Pg%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                               %pg 为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
%%%%%%%%%%%%%主循环,按照公式依次迭代%%%%%%%%%%%%%%%%
for t = 1:M
    for i = 1:N
        v(i,:) = w * v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        fx(i) = fitness(x(i,:));
        if fx(i)<p(i)
            p(i) = fx(i);
            y(i,:) = x(i,:);
        end
        if p(i)<fitness(pg)
            pg = y(i,:);
        end
    end
```

```
    [sortf,sortx] = sort(fx);
    exIndex = round((N - 1)/2);
    x(sortx((N - exIndex + 1):N)) = x(sortx(1:exIndex));          % 替换位置
    v(sortx((N - exIndex + 1):N)) = v(sortx(1:exIndex));          % 替换速度
end
xm = pg';
fv = fitness(pg);
```

【例 3-6】 使用基于自然选择的混合粒子群算法，求解函数

$$f(x) = 1\Big/\left[0.3 + \sum_{i=1}^{5} \frac{i-1}{(x_i+1)^2}\right]$$

的最小值，其中$-10 \leqslant x_i \leqslant 10$。粒子数为 50，学习因子均为 2，惯性权重为 0.9，迭代步数为 1000。

解 首先建立目标函数代码如下：

```
function y = natureFunc(x)
y = 0;
for i = 1:5
    y = y + (i - 1)/((x(i) + 1)^2)
end
y = 1/(0.3 + y)
```

在 MATLAB 命令行窗口输入如下代码：

```
[xm,fv] = PSO_nature(@natureFunc,50,2,2,0.8,100,5)
```

运行后得到结果为：

```
>> xm =
 - 12.574754662564919
    0.346363484290835
   - 1.000001495407084
    1.626759539954806
    1.000552183389289
fv =
     1.118121173014267e - 12
```

以上结果表明，基于杂交的混合粒子群算法精度也非常高。

3.5.3 基于免疫的粒子群算法

基于免疫的粒子群算法是在免疫算法的基础上采用粒子群优化对抗体群体进行更新，可以解决免疫算法收敛速度慢的缺点。

基于免疫的混合粒子群算法步骤如下。

(1) 确定学习因子 c_1 和 c_2、粒子(抗体)群体个数 M。

(2) 由 logistic 回归分析映射产生 M 个粒子(抗体)x_i及其速度 v_i,其中 $i=1,2,\cdots,N$,最后形成初始粒子(抗体)群体 P_0。

(3) 生产免疫记忆粒子(抗体)。计算当前粒子(抗体)群体 P 中粒子(抗体)的适应值并判断算法是否满足结束条件,如果满足则结束并输出结果,否则继续运行。

(4) 更新局部和全局最优解,并根据下面公式更新粒子位置和速度:

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1,2,\cdots,d$$

$$v_{i,j}(t+1) = w \cdot v_{i,j}(t) + c_1 r_1[p_{i,j} - x_{i,j}(t)] + c_2 r_2[p_{g,j} - x_{i,j}(t)]$$

(5) 由 logistic 映射产生 N 个新的粒子(抗体)。

(6) 基于浓度的粒子(抗体)选择。用群体中相似抗体百分比计算生产 $N+M$ 个新粒子(抗体)的概率,依照概率大小选择 N 个粒子(抗体)形成粒子(抗体)群 P,然后转入第(3)步。

算法流程图如图 3-7 所示。

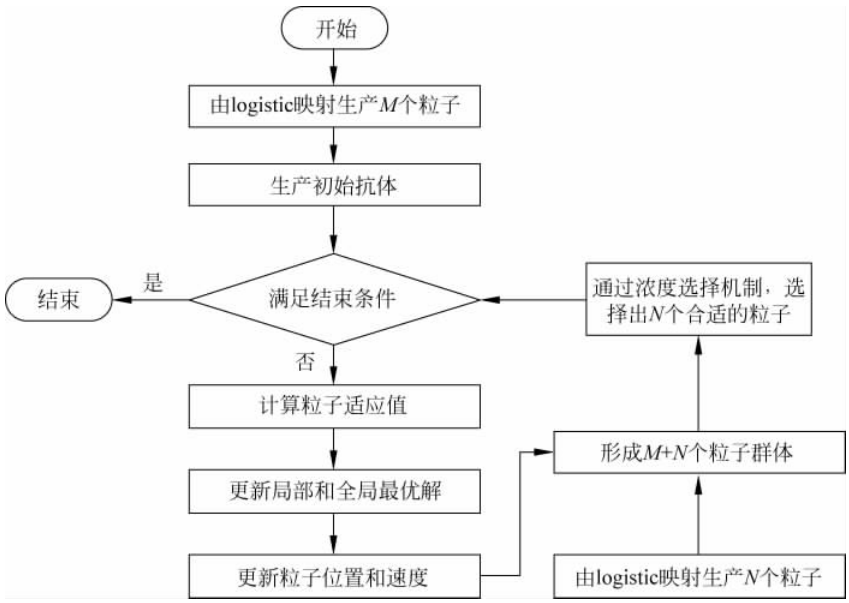

图 3-7 免疫粒子群算法流程

将实现自适应权重的优化函数命名为 PSO_immu,在 MATLAB 中编写实现以上步骤的代码如下:

```
function [x,y,Result] = PSO_immu(func,N,c1,c2,w,MaxDT,D,eps,DS,replaceP,minD,Psum)
format long;
%%%%%% 给定初始化条件 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% c1 = 2;                    % 学习因子 1
% c2 = 2;                    % 学习因子 2
% w = 0.8;                   % 惯性权重
% MaxDT = 100;               % 最大迭代次数
% D = 2;                     % 搜索空间维数(未知数个数)
% N = 100;                   % 初始化群体个体数目
```

```
% eps = 10^(-10);                    % 设置精度(在已知最小值时候用)
% DS = 8;                            % 每隔 DS 次循环就检查最优个体是否变优
% replaceP = 0.5;                    % 粒子的概率大于 replaceP 将被免疫替换
% minD = 1e-10;                      % 粒子间的最小距离
% Psum = 0;                          % 个体最佳的和
range = 100;
count = 0;
%%%%%% 初始化种群的个体 %%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = -range + 2*range*rand;      % 随机初始化位置
        v(i,j) = randn;                      % 随机初始化速度
    end
end

%%%% 先计算各个粒子的适应度,并初始化 Pi 和 Pg %%%%%%%%%%%%%%%%%%%%
for i = 1:N
    p(i) = feval(func,x(i,:));
    y(i,:) = x(i,:);
end
pg = x(1,:);                         % pg 为全局最优
for i = 2:N
    if feval(func,x(i,:))<feval(func,pg)
        pg = x(i,:);
    end
end
%%%%%%%%%%%%%% 主循环,按照公式依次迭代,直到满足精度要求 %%%%%%%%%
for t = 1:MaxDT
    for i = 1:N
        v(i,:) = w*v(i,:) + c1*rand*(y(i,:) - x(i,:)) + c2*rand*(pg - x(i,:));
        x(i,:) = x(i,:) + v(i,:);
        if feval(func,x(i,:))<p(i)
            p(i) = feval(func,x(i,:));
            y(i,:) = x(i,:);
        end
        if p(i)<feval(func,pg)
            pg = y(i,:);
            subplot(1,2,1);
            bar(pg,0.25);
            axis([0 3 -40 40]) ;
            title(['Iteration ', num2str(t)]); pause(0.1);
            subplot(1,2,2);
          plot(pg(1,1),pg(1,2),'rs','MarkerFaceColor','r', 'MarkerSize',8)
            hold on;
            plot(x(:,1),x(:,2),'k.');
            set(gca,'Color','g')
            hold off;
            grid on;
            axis([-100 100 -100 100]);
```

```
            title(['Global Min = ',num2str(p(i))]);
            xlabel(['Min_x = ',num2str(pg(1,1)),'Min_y = ',num2str(pg(1,2))]);

        end
    end
    Pbest(t) = feval(func,pg) ;
%     if Foxhole(pg,D)< eps                          %如果结果满足精度要求则跳出循环
%         break;
%     end
%%%%% 开始进行免疫 %%%%%%%%%%%%%%%%%%
    if t > DS
       if mod(t,DS) == 0 && (Pbest(t - DS + 1) - Pbest(t))< 1e - 020     %如果连续 DS 代数,
                                                  %群体中的最优没有明显变优,则进行免疫
            %在函数测试的过程中发现,经过一定代数的更新,个体最优不完全相等,但变化非常小,
           for i = 1:N                             %先计算出个体最优的和
             Psum = Psum + p(i);
           end

           for i = 1:N                             %免疫程序

               for j = 1:N                         %计算每个个体与个体 i 的距离
                   distance(j) = abs(p(j) - p(i));
               end
               num = 0;
               for j = 1:N                         %计算与第 i 个个体距离小于 minD 的个数
                   if distance(j)< minD
                       num = num + 1;
                   end
               end
               PF(i) = p(N - i + 1)/Psum;          %计算适应度概率
               PD(i) = num/N;                      %计算个体浓度

               a = rand;                           %随机生成计算替换概率的因子
               PR(i) = a * PF(i) + (1 - a) * PD(i);  %计算替换概率
           end

            for i = 1:N
                if PR(i)> replaceP
                    x(i,:) = - range + 2 * range * rand(1,D);
               count = count + 1;
                end
           end
       end
    end
end
```

```
%%%%%%%% 最后给出计算结果 %%%%%%%%%%%%%%%%%%%%%%
x = pg(1,1);
y = pg(1,2);
Result = feval(func,pg);
%%%%%%%%%%% 算法结束 %%%%%%%%%%%%%%%%%%%%
function probability(N,i)
PF = p(N - i)/Psum;                          % 适应度概率
disp(PF);
for jj = 1:N
  distance(jj) = abs(P(jj) - P(i));
end
num = 0;
for ii = 1:N
    if distance(ii)< minD
        num = num + 1;
    end
end
PD = num/N;                                  % 个体浓度
PR = a * PF + (1 - a) * PD;                  % 替换概率
```

【例 3-7】 使用基于模拟退火的混合粒子群算法，求解函数

$$f(x)=\frac{\cos\sqrt{x_1^2+x_2^2}-1}{[1+(x_1^2-x_2^2)]^2}+0.5$$

的最小值，其中$-10\leqslant x_i\leqslant 10$。粒子数为 50，学习因子均为 2，退火常数为 0.6，迭代步数为 1000。

解 首先建立目标函数代码如下：

```
function y = immuFunc(x)
   y = (cos(x(1)^2 + x(2)^2) - 1)/((1 + (x(1)^2 - x(2)^2))^2) + 0.5;
end
```

在 MATLAB 命令行窗口输入如下代码：

```
[xm,fv] = PSO_immu (@immuFunc,50,2,2,0.8,100,5,0.0000001,10,0.6,0.000000000000000001,0)
```

运行后得到结果为：

```
>> xm =
   1.139888959718036
fv =
  - 1.515992561220435
```

得到目标函数取最小值时的自变量 xm 变化图如图 3-8 所示。

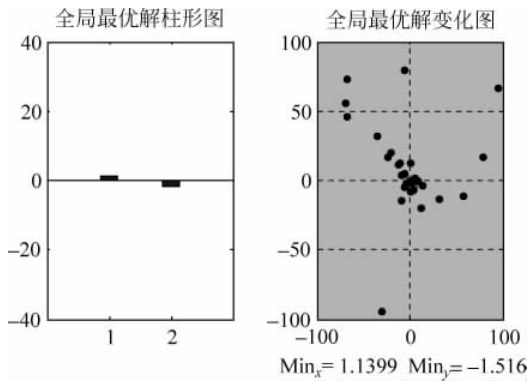

图 3-8 目标函数取最小值时的自变量 xm 变化图

3.5.4 基于模拟退火的算法

基于模拟退火的算法在搜索过程中具有突跳的能力，可以有效地避免搜索陷入局部极小解。

基于杂交的混合粒子群算法步骤如下。

(1) 随机设置各个粒子的速度和位置。

(2) 评价评价每个粒子的适应度，将粒子的位置和适应值存储在粒子的个体极值 p_{best} 中，将所有 p_{best} 中最优适应值的个体位置和适应值保存在全局极值 g_{best} 中。

(3) 确定初始温度。

(4) 根据下式确定当前温度下各粒子 p_i 的适应值：

$$\text{TF}(p_i) = \frac{e^{-(f(p_i)-f(p_g))/t}}{\sum_{i=1}^{N} e^{-(f(p_i)-f(p_g))/t}}$$

(5) 从所有 p_i 中确定全局最优的替代值 p_i'，并根据下面两个公式更新各粒子的位置和速度。

$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1), \quad j = 1,2,\cdots,d$$

$$v_{i,j}(t+1) = \varphi\{v_{i,j}(t) + c_1 r_1[p_{i,j} - x_{i,j}(t)] + c_2 r_2[p_{g,j} - x_{i,j}(t)]\}$$

$$\varphi = \frac{2}{2-(c_1+c_2)-\sqrt{(c_1+c_2)^2-4(c_1+c_2)}}$$

(6) 计算粒子目标值，并更新 p_{best} 和 g_{best}，然后进行退温操作。

(7) 当算法达到其停止条件，则停止搜索并输出结果；否则返回到第(4)步继续搜索。

(8) 初始温度和退温方式对算法有一定的影响，一般采用如下所示的初始温度和退温方式：

$$t_{k+1} = \lambda t_k, \quad t_0 = f(p_g)/\text{In}5$$

将实现自适应权重的优化函数命名为 PSO_lamda，在 MATLAB 中编写实现以上步骤的代码如下：

```
function [xm,fv] = PSO_lamda(fitness,N,c1,c2,lamda,M,D)
format long;
%N初始化群体个体数目
%c1学习因子1
%c2学习因子2
%lamda退火常数惯性权重
%M最大迭代次数
%D搜索空间维数
%%%%%%%%%%%%初始化种群的个体%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:N
    for j = 1:D
        x(i,j) = randn;                          %初始化位置
        v(i,j) = randn;                          %初始化速度
    end
end
%%%%%%%%%%先计算各个粒子的适应度,并初始化Pi和Pg%%%%%%%%%%
for i = 1:N
    p(i) = fitness(x(i,:));
    y(i,:) = x(i,:);
end
pg = x(N,:);                                     %pg为全局最优
for i = 1:(N-1)
    if fitness(x(i,:))<fitness(pg)
        pg = x(i,:);
    end
end
%%%%%主循环,按照公式依次迭代%%%%%%%%%%%%%%%%%%%%%%%%%%%%
T = - fitness(pg)/log(0.2);
for t = 1:M
    groupFit = fitness(pg);
    for i = 1:N
        Tfit(i) = exp( -(p(i) - groupFit)/T);
    end
    SumTfit = sum(Tfit);
    Tfit = Tfit/SumTfit;
    pBet = rand();
    for i = 1:N
        ComFit(i) = sum(Tfit(1:i));
        if pBet <= ComFit(i)
            pg_plus = x(i,:);
           break;
        end
    end
    C = c1 + c2;
    ksi = 2/abs(2 - C - sqrt(C^2 - 4 * C));
    for i = 1:N
        v(i,:) = ksi * (v(i,:) + c1 * rand * (y(i,:) - x(i,:)) + c2 * rand * (pg_plus -
x(i,:)));
```

```
            x(i,:) = x(i,:) + v(i,:);
            if fitness(x(i,:))< p(i)
                p(i) = fitness(x(i,:));
                y(i,:) = x(i,:);
            end
            if p(i)< fitness(pg)
                pg = y(i,:);
            end
        end
        T = T * lamda;
        Pbest(t) = fitness(pg);
    end
    xm = pg';
    fv = fitness(pg);
```

【例 3-8】 使用基于模拟退火的混合粒子群算法，求解函数

$$f(x)=1\Big/\left[0.7+\sum_{i=1}^{5}\frac{i+2}{(x_i-1)^2+0.5}\right]$$

的最小值，其中$-10\leqslant x_i\leqslant 10$。粒子数为 50，学习因子均为 2，退火常数为 0.6，迭代步数为 1000。

解 首先建立目标函数代码如下：

```
function y = lamdaFunc(x)
y = 0;
for i = 1:5
    y = y + (i + 2)/(((x(i) - 1)^2) + 0.5);
end
y = 1/(0.7 + y);
```

在 MATLAB 命令行窗口输入代码：

```
[xm,fv] = PSO_lamda(@lamdaFunc,50,2,2,0.5,100,5)
```

运行后得到结果为：

```
>> xm =
   1.814532522705427
   1.669344110734507
   0.912442817927282
   1.045716794351236
   0.837420691707699
fv =
   0.023477527185062
```

以上结果表明，基于杂交的混合粒子群算法精度也非常高。

3.6 本章小结

粒子群算法思想简单,使用方便。本章首先介绍了粒子群算法的基础,包括其研究内容、特点和应用领域等。随后对基本粒子群算法的原理、流程等内容做了详细的介绍,并介绍了三种权重改进粒子群算法。最后,针对粒子群不同的混合对象,举例说明了粒子群算法的各种混合应用。