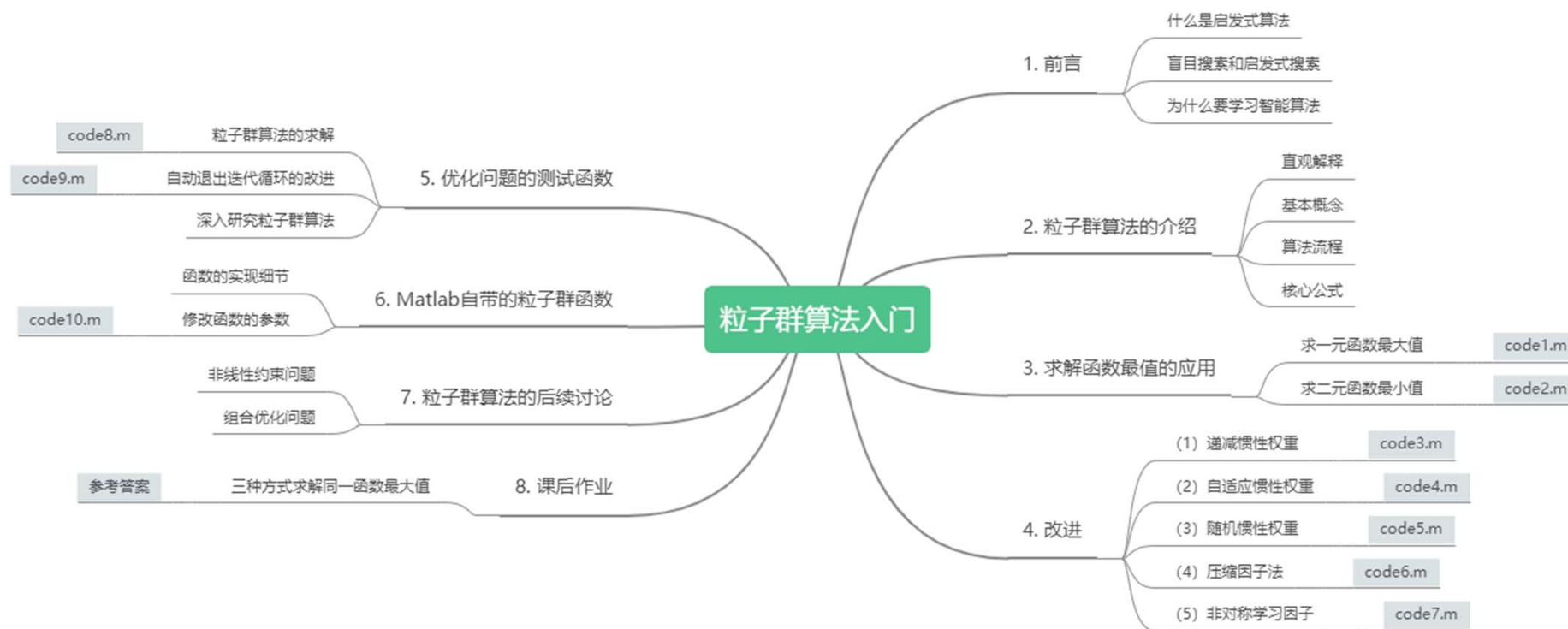


粒子群算法入门



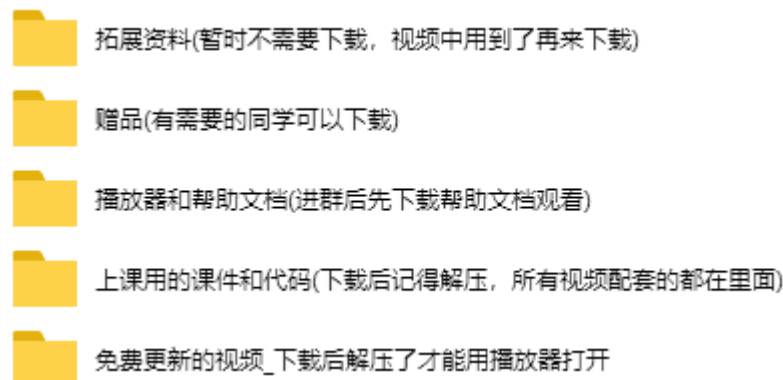
这里用的绘图工具是在线的: <https://www.processon.com/>

本节代码参考: 我要自学网的龚飞老师《Matlab2016数值计算与智能算法》

关注微信公众号"数学建模学习交流"获取更多优质资料

温馨提示

- (1) 视频中提到的附件可在**售后群的群文件**中下载。
包括**讲义、代码、我视频中推荐的资料**等。



(2) 关注我的**微信公众号《数学建模学习交流》**，后台发送**“软件”**两个字，可获得常见的建模软件下载方法；发送**“数据”**两个字，可获得建模数据的获取方法；发送**“画图”**两个字，可获得数学建模中常见的画图方法。另外，也可以看看公众号的历史文章，里面发布的都是对大家有帮助的技巧。

(3) **购买更多优质精选的数学建模资料**，可关注我的微信公众号《数学建模学习交流》，在后台发送**“买”**这个字即可进入店铺进行购买。

(4) 视频价格不贵，但价值很高。单人购买观看只需要**58元**，和另外两名队友一起购买人均仅需**46元**，视频本身也是下载到本地观看的，所以请大家**不要侵犯知识产权**，对视频或者资料进行二次销售。

粒子群算法入门

数模比赛中, 经常见到有同学“套用”启发式算法(数模中常称为智能优化算法) 去求解一些数模问题, 事实上, 很大一部分问题是不需要用到启发式算法求解的, Matlab中内置的函数足够我们使用了。但是如果遇到的优化问题特别复杂的话, 启发式算法就是我们求解问题的一大法宝。

今天我们就来学习第一个智能优化算法: 粒子群算法, 其全称为粒子群优化算法(Particle Swarm Optimization, PSO)。它是通过模拟鸟群觅食行为而发展起来的一种基于群体协作的搜索算法。本节我们主要侧重学习其思想, 并将其用于求解函数的最值问题, 下一节我们会使用粒子群算法求解几类较难处理的优化类问题。

温馨提示: 学习本节前请先自学更新10蒙特卡罗模拟和更新12数学规划模型。

什么是启发式算法?

注: “智能算法”是指在工程实践中提出的一些比较“新颖”的算法或理论, 因此智能算法的范围要比启发式算法更大一点, 如果某种智能算法可以用来解决优化问题, 那么这种算法也可能认为是启发式算法。

启发式算法百度百科上的定义: 一个基于直观或经验构造的算法, 在**可接受的花费**下给出待解决**优化问题**的一个**可行解**。

(1) 什么是可接受的花费?

计算时间和空间能接受 (求解一个问题要几万年 or 一万台电脑)

(2) 什么是优化问题?

工程设计中优化问题 (optimization problem) 指在一定约束条件下, 求解一个目标函数的最大值(或最小值)问题。

注: 实际上和我们之前学过的规划问题的定义一样, 称呼不同而已。

(3) 什么是可行解?

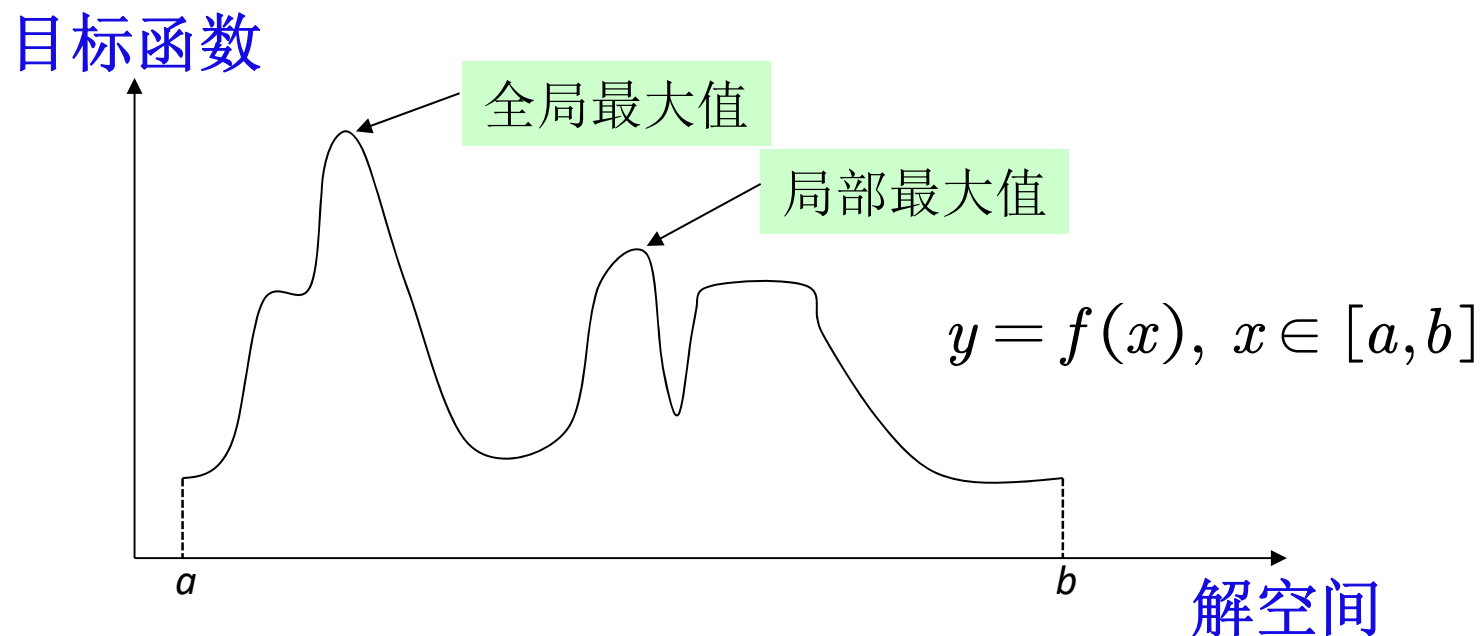
得到的结果能用于工程实践 (不一定非要是最优解)

常见的启发式算法:

粒子群、模拟退火、遗传算法、蚁群算法、禁忌搜索算法等等
(启发式算法解决的问题大同小异, 只要**前三个算法**学会了在数学建模中就足够了)

注意: 在不引起误会的情况下, 以后我们不区分启发式算法和智能算法。


最简单的优化问题



思考: 怎么找到这个一元函数的最大值?
(只有一个上下界约束, 即函数的定义域)

盲目搜索 和 启发式搜索

参考资料: <https://wenku.baidu.com/view/3c42f099974bcf84b9d528ea81c758f5f61f29c1.html>

 数学建模学习交流

关注微信公众号"数学建模学习交流"获取更多优质资料

盲目搜索

假设图中的 $a = 1, b = 10$, 我们要找出连续函数 $y = f(x)$ 在 $[1, 10]$ 的最大值。

(1) 枚举法

取 $x_1 = 1, x_2 = 1.0001, x_3 = 1.0002, \dots, x_{9001} = 10$

分别计算 $f(x_1), f(x_2), \dots, f(x_{9001})$, 看其中哪个函数值最大。

(如果不考虑计算时间, 我们可以划分的更小, 这样可以增加计算精度)

(2) 蒙特卡罗模拟

随机在 $[a, b]$ 区间上取 N 个样本点, 即 x_1, x_2, \dots, x_N 都是在 $[a, b]$ 取的随机数

分别计算 $f(x_1), f(x_2), \dots, f(x_N)$, 看其中哪个函数值最大。

(如果不考虑计算时间, 我们可以取更多的样本点, 这样可以增加计算精度)

有什么问题?

如果现在要求最值的函数是一个多变量的函数, 例如是一个10个变量的函数, 那么就完蛋了! (要考虑的情况随着变量数呈指数增长, 计算时间肯定不够)

启发式搜索

盲目搜索还是启发式搜索?

按照预定的策略实行搜索, 在搜索过程中获取的中间信息**不用来**改进策略, 称为盲目搜索; 反之, **如果利用了中间信息来改进搜索策略则称为启发式搜索。**

例如: 蒙特卡罗模拟用来求解优化问题就是盲目搜索, 还有大家熟悉的枚举法也是盲目搜索。

• 关于“启发式”, 可以简单总结为下面两点:

- 1) 任何有助于找到问题的最优解, 但不能保证找到最优解的方法均是启发式方法;
- 2) 有助于加速求解过程和找到较优解的方法是启发式方法。

为什么要学习启发式算法?

从“比赛”的角度出发:

- (1) TSP(旅行商问题)这类组合优化问题
- (2) 非线性整数规划问题 (例如书店买书问题)

之前学过的蒙特卡罗模拟实际上是随机找解来尝试, 如果解集中存在的元素特别多, 那么效果就不理想。

从“学习”的角度出发:

- (1) 训练最优化的思维
- (2) 提高自身的编程水平
- (3) 各个专业都有广泛应用

粒子群算法



1995年, 美国学者Kennedy和Eberhart共同提出了粒子群算法, 其基本思想源于对鸟类群体行为进行建模与仿真的研究结果的启发。

它的核心思想是利用群体中的**个体对信息的共享**使整个群体的运动在问题求解空间中产生从无序到有序的演化过程, 从而获得问题的可行解。

最初提出的论文: Kennedy J, Eberhart R. Particle swarm optimization[C]// Proceedings of ICNN'95 - International Conference on Neural Networks. IEEE, 1995.

粒子群算法的直观解释

设想这样一个场景：一群鸟在搜索食物

假设：

- (1) 所有的鸟都不知道食物在哪
- (2) 它们知道自己的当前位置距离食物有多远
- (3) 它们知道离食物最近的鸟的位置

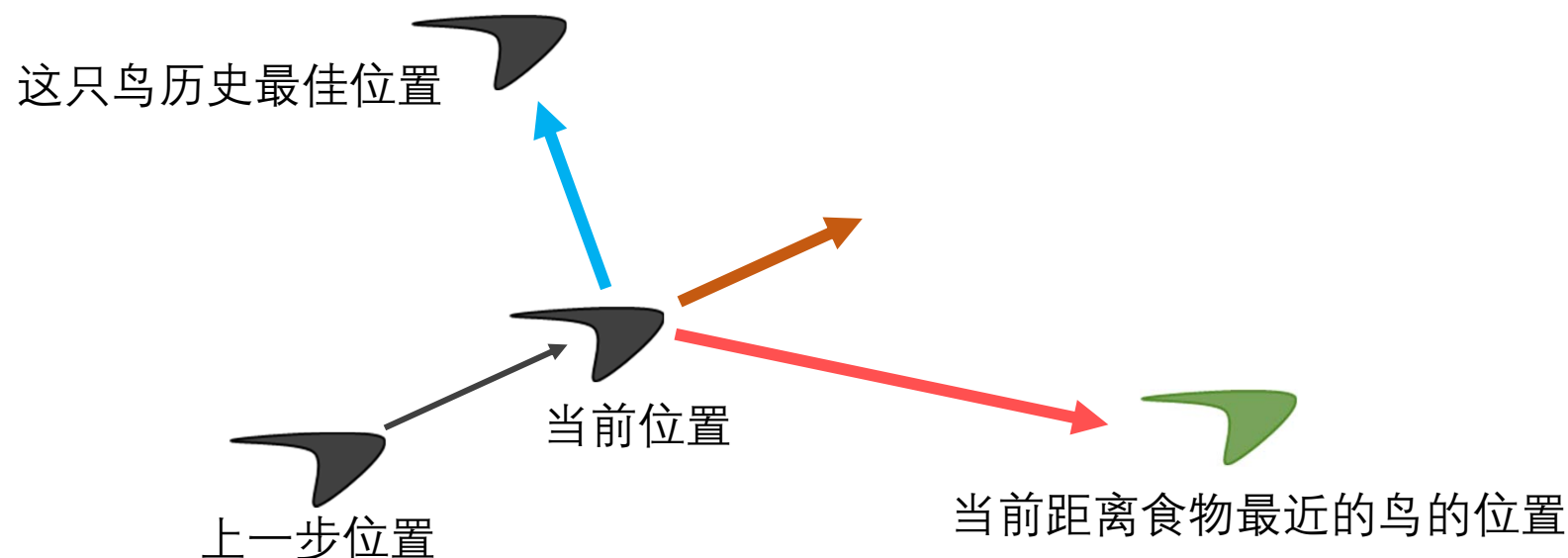
那么想一下这时候会发生什么？

粒子群算法的直观解释

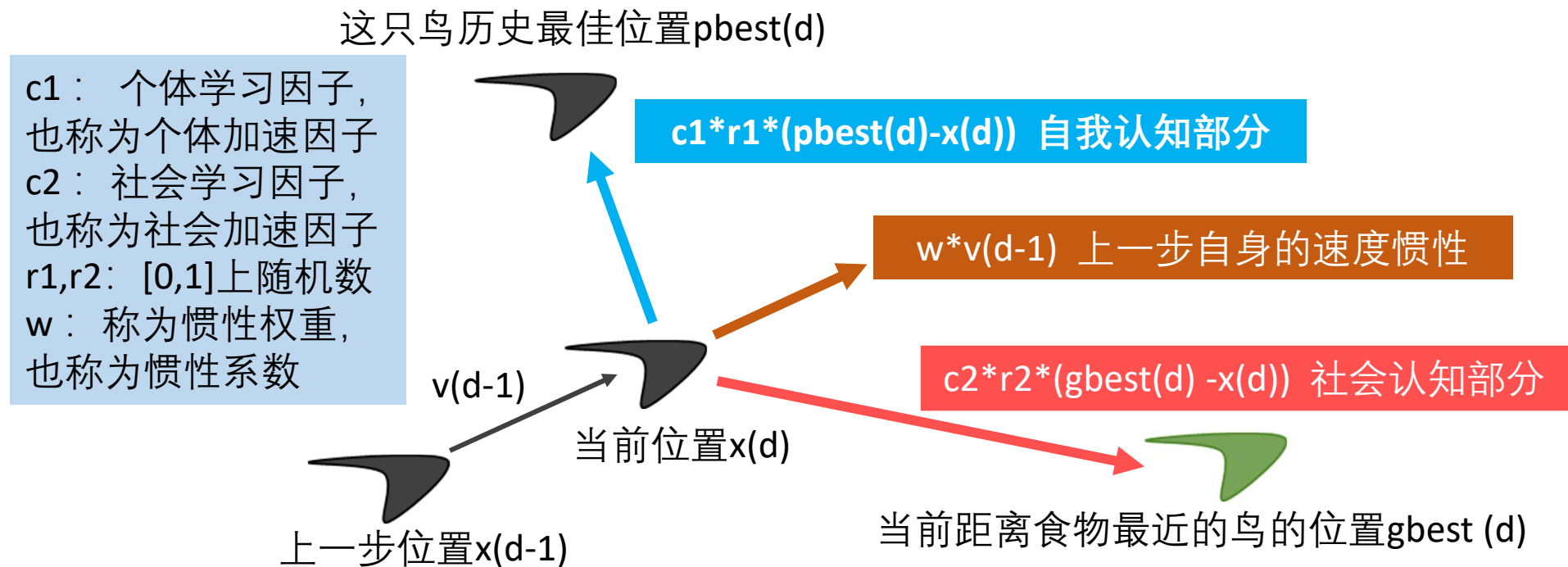
首先, 离食物最近的鸟会对其他的鸟说: 兄弟们, 你们快往我这个方向来, 我这离食物最近;

与此同时, 每只鸟在搜索食物的过程中, 它们的位置也在不停变化, 因此每只鸟也知道自己离食物最近的位置, 这也是它们的一个参考;

最后, 鸟在飞行中还需要考虑一个惯性。



图形的进一步解释



这只鸟第 d 步所在的位置 = 第 $d-1$ 步所在的位置 + 第 $d-1$ 步的速度 * 运动的时间

$$x(d) = x(d-1) + v(d-1) * t \quad (\text{每一步运动的时间 } t \text{ 一般取 } 1)$$

这只鸟第 d 步的速度 = 上一步自身的速度惯性 + 自我认知部分 + 社会认知部分

$$v(d) = w*v(d-1) + c1*r1*(pbest(d)-x(d)) + c2*r2*(gbest(d)-x(d)) \quad (\text{三个部分的和})$$

粒子群算法中的基本概念

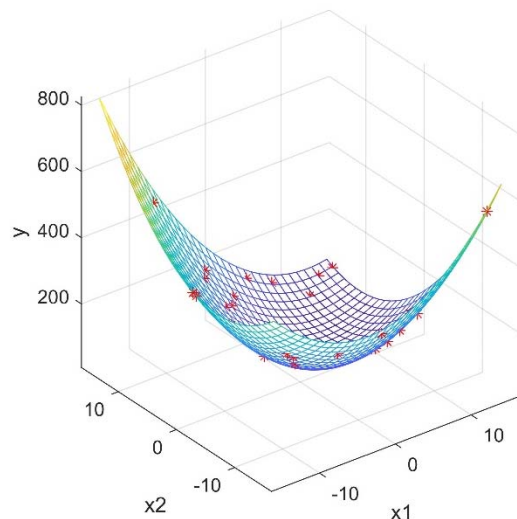
- **粒子**: 优化问题的候选解
- **位置**: 候选解所在的位置
- **速度**: 候选解移动的速度
- **适应度**: 评价粒子优劣的值, 一般设置为目标函数值
- **个体最佳位置**: 单个粒子迄今为止找到的最佳位置
- **群体最佳位置**: 所有粒子迄今为止找到的最佳位置

这里用Matlab演示粒子群算法的计算过程

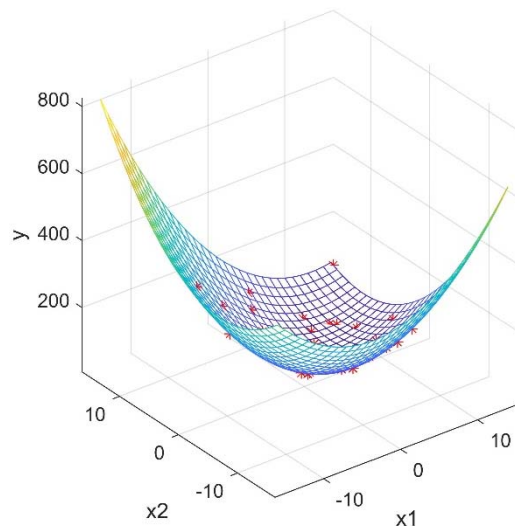
粒子群寻找最小值ing

$$y = x_1^2 + x_2^2 - x_1x_2 - 10x_1 - 4x_2 + 60$$

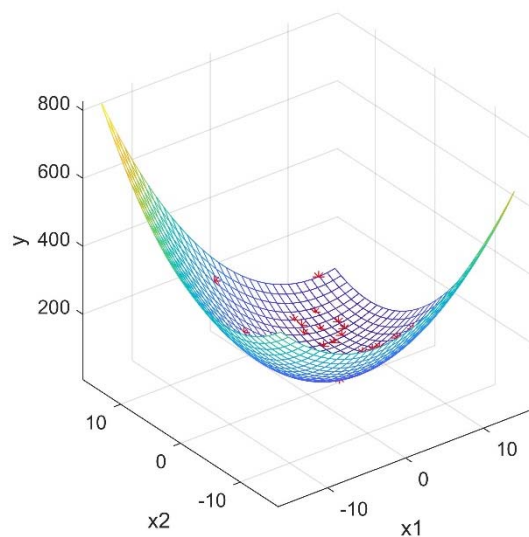
迭代1次



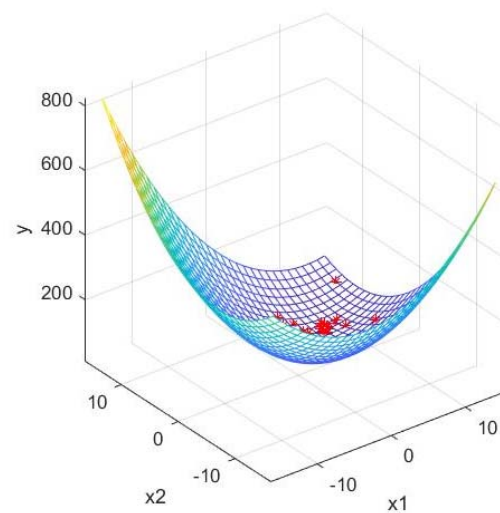
迭代2次



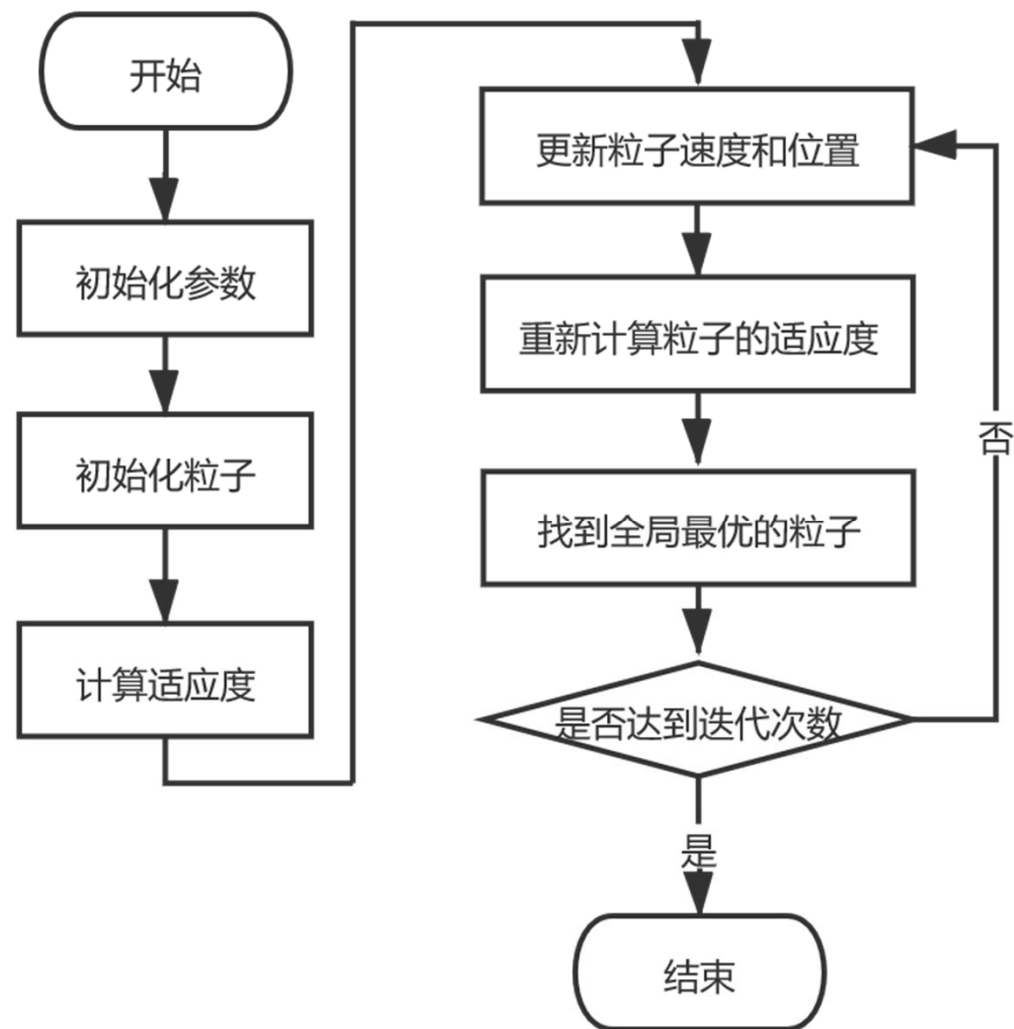
迭代3次



迭代100次



粒子群算法流程图



这里用的绘图工具是在线的：<https://www.processon.com/>

符号说明

符号	含义
n	粒子个数
c_1	粒子的个体学习因子, 也称为个体加速因子
c_2	粒子的社会学习因子, 也称为社会加速因子
w	速度的惯性权重
v_i^d	第d次迭代时, 第i个粒子的速度
x_i^d	第d次迭代时, 第i个粒子所在的位置
$f(x)$	在位置x时的适应度值(一般取目标函数值)
$pbest_i^d$	到第d次迭代为止, 第i个粒子经过的最好的位置
$gbest^d$	到第d次迭代为止, 所有粒子经过的最好的位置

我用的公式编辑器是Axmath: <http://www.amyxun.com/>

粒子群算法的核心公式

这只鸟第d步的速度 = 上一步自身的速度惯性 + 自我认知部分 + 社会认知部分
 $v(d) = w*v(d-1) + c1*r1*(pbest(d)-x(d)) + c2*r2*(gbest(d)-x(d))$ (三个部分的和)

$$v_i^d = wv_i^{d-1} + c_1r_1(pbest_i^d - x_i^d) + c_2r_2(gbest^d - x_i^d)$$

其中 r_1, r_2 是 $[0, 1]$ 上的随机数

注意, 这里我们没有在变量说明的表格中放入 r_1 和 r_2 这两个随机数, 是因为他们表示的含义不太重要, 我们只需要简单的交代一下就行。

这只鸟第d+1步所在的位置 = 第d步所在的位置 + 第d步的速度 * 运动的时间
 $x(d+1) = x(d) + v(d) * t$ (每一步运动的时间 t 一般取1)

$$x_i^{d+1} = x_i^d + v_i^d$$

预设参数: 学习因子

$$v_i^d = wv_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)$$

It was soon realized that there is no good way to guess whether *p*- or *g*-increment should be larger. Thus, these terms were also stripped out of the algorithm. The stochastic factor was multiplied by 2 to give it a mean of 1, so that agents would “overfly” the target about half the time. This version outperforms the previous versions. Further research will show whether there is an optimum value for the constant currently set at 2, whether the value should be evolved for each problem, or whether the value can be determined from some knowledge of a particular problem. The current simplified particle swarm optimizer now adjusts velocities by the following formula:

$$\begin{aligned} vx[[[]]] &= vx[[[]]] + \\ &2 * rand() * (pbestx[[[]]] - presentx[[[]]]) + \\ &2 * rand() * (pbestx[[[]]][gbest] - presentx[[[]]]) \end{aligned}$$

在最初提出粒子群算法的论文中指出, 个体学习因子和社会(或群体)学习因子取2比较合适。

(注意: 最初提出粒子群算法的这篇论文没有惯性权重)

最初提出的论文: Kennedy J, Eberhart R. Particle swarm optimization[C]// Proceedings of ICNN'95 - International Conference on Neural Networks. IEEE, 1995.

预设参数: 惯性权重

$$v_i^d = w v_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)$$

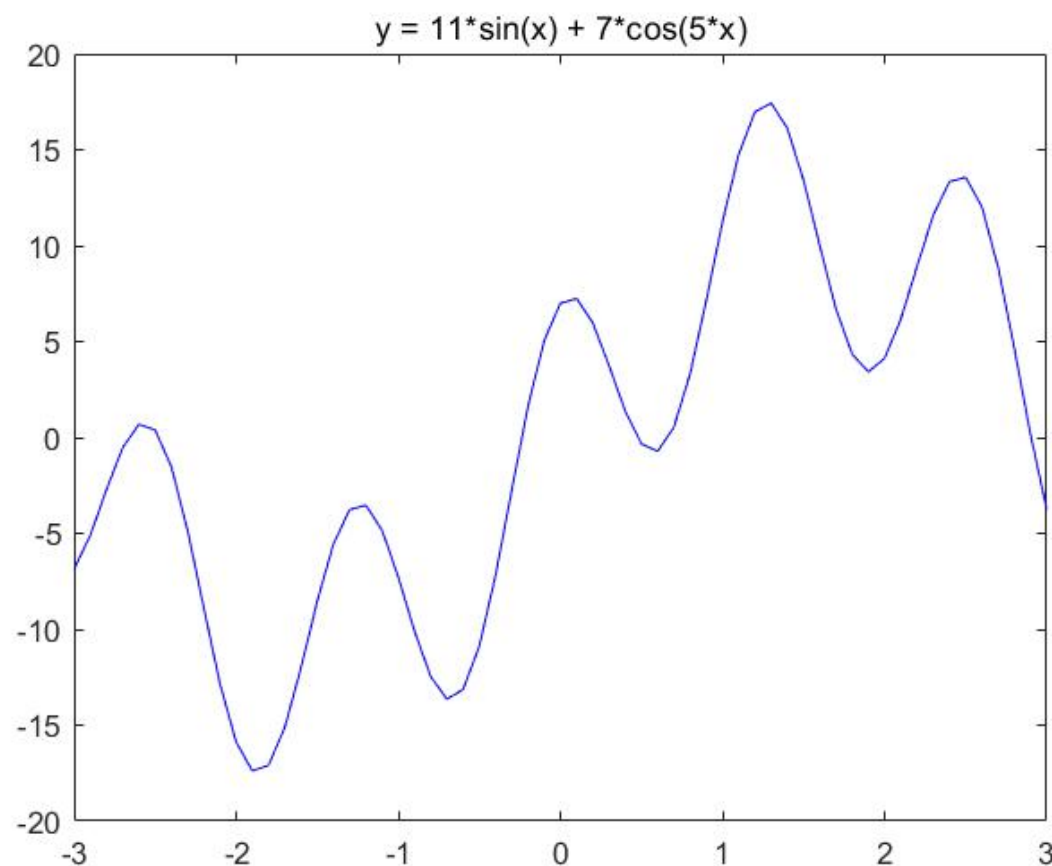
In this paper, we have introduced a parameter *inertia weight* into the original particle swarm optimizer. Simulations have been performed to illustrate the impact of this parameter on the performance of PSO. It is concluded that the PSO with the inertia weight in the range [0.9, 1.2] on average will have a better performance; that is, it has a bigger chance to find the global optimum within a reasonable number of iterations. Furthermore, a time decreasing inertia weight is introduced which brings in a significant improvement on the PSO performance. Simulations have been done to support it.

论文中得到的结论: 惯性权重取0.9-1.2是比较合适的, 一般取0.9就行

引入惯性权重的论文: SHI, Y. A Modified Particle Swarm Optimizer[C]// Proc. of IEEE ICEC conference, Anchorage. 1998.

例1: 求一元函数的最大值

求函数 $y = 11\sin x + 7\cos(5x)$ 在 $[-3, 3]$ 内的最大值



用我们之前学过的函数解决

打开code0.m代码演示

```
x0 = 0;  
A=[]; b=[];  
Aeq=[]; beq=[];  
x_lb = -3; % x的下界  
x_ub = 3; % x的上界  
[x,fval] = fmincon(@Obj_fun1,x0,A,b,Aeq,beq,x_lb,x_ub)  
fval = -fval
```

```
function y = Obj_fun1(x)  
% 如果调用fmincon函数, 则需要添加负号改为求最小值  
y = -(11*sin(x) + 7*cos(5*x));  
end
```

```
x =  
  
1.2750  
  
fval =  
  
17.4928
```

测试：如果把初始值改为 $x_0 = 2$ ，结果会是什么？

$x = 2.4638$, $fval = 13.6847$ (局部最大值)

设置粒子群算法的参数

打开code1.m代码演示

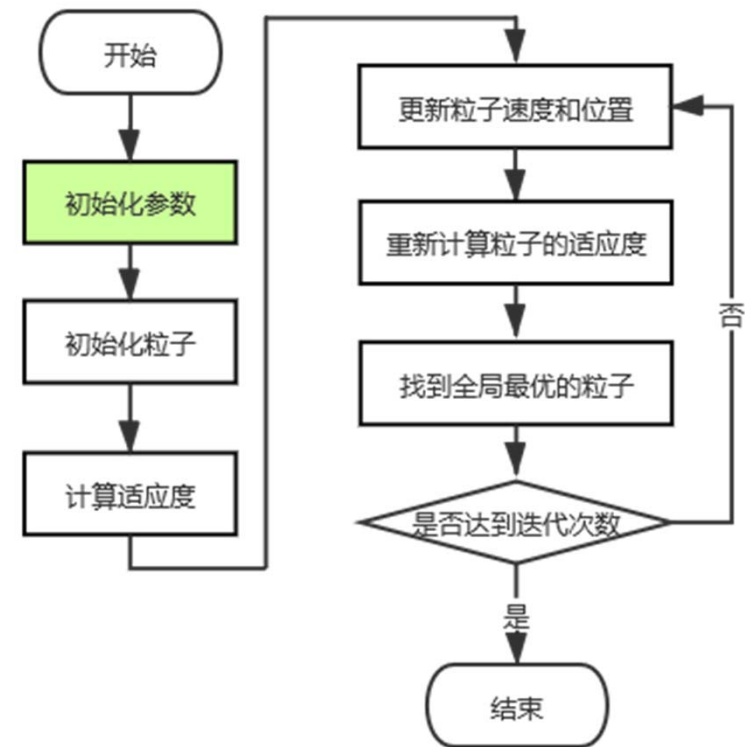
```
n = 10; % 粒子数量
narvs = 1; % 变量个数(函数中有几个自变量)
c1 = 2; % 每个粒子的个体学习因子
c2 = 2; % 每个粒子的社会学习因子
w = 0.9; % 惯性权重
K = 50; % 迭代的次数
vmax = 1.2; % 粒子的最大速度
x_lb = -3; % x的下界
x_ub = 3; % x的上界
```

- (1) 增加粒子数量会增加我们找到更好结果的可能性, 但会增加运算的时间。
- (2) $c1, c2, w$ 这三个量有很多改进空间, 未来我再来专门讲怎么去调整。
- (3) 迭代的次数 K 越大越好吗? 不一定哦, 如果现在已经找到最优解了, 再增加迭代次数是没有意义的。
- (4) 这里出现了粒子的最大速度, 是为了保证下一步的位置离当前位置别太远, 一般取自变量可行域的10%至20%(不同文献看法不同)。
- (5) x 的下界和上界是保证粒子不会飞出定义域。

注意: 假设目标函数是二元函数, 且 x_1 和 x_2 都位于-3至3之间, 则:

(1) $\text{narvs} = 2$

(2) $x_lb = [-3 \ -3]; x_ub = [3 \ 3]; v_{\max} = [1.2 \ 1.2]$



初始化粒子的位置和速度

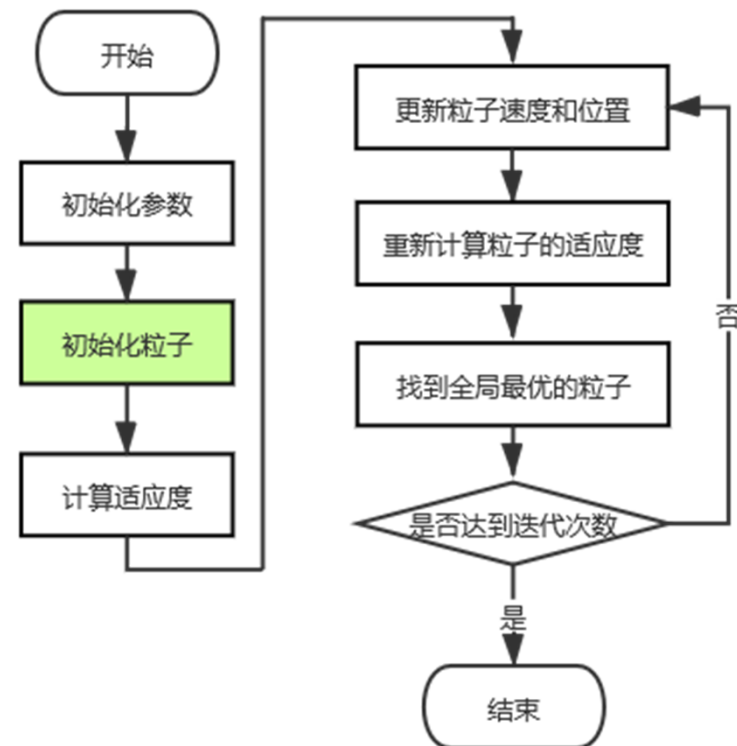
```
x = zeros(n,narvs);
for i = 1: narvs
    % 随机初始化粒子所在的位置在定义域内
    x(:,i) = x_lb(i) + (x_ub(i)-x_lb(i))*rand(n,1);
end
% 随机初始化粒子的速度, 设置为[-vmax,vmax]
v = -vmax + 2*vmax .* rand(n,narvs);
```

- (1) 这里为什么要写成下标的形式?
保证程序的兼容性, “适配”多元函数
- (2) 行向量为什么可以和矩阵来点乘?

```
ans =
>> [1 2] .* [3,4;5,6;7,8]
     3     8
     5    12
     7    16
```

- (3) 行向量为什么可以和矩阵来相加?

```
ans =
>> [1 2] + [3,4;5,6;7,8]
     4     6
     6     8
     8    10
```



注意: 这种写法只支持2017及之后的Matlab, 老版本的同学请自己使用repmat函数将向量扩充为矩阵后再运算。

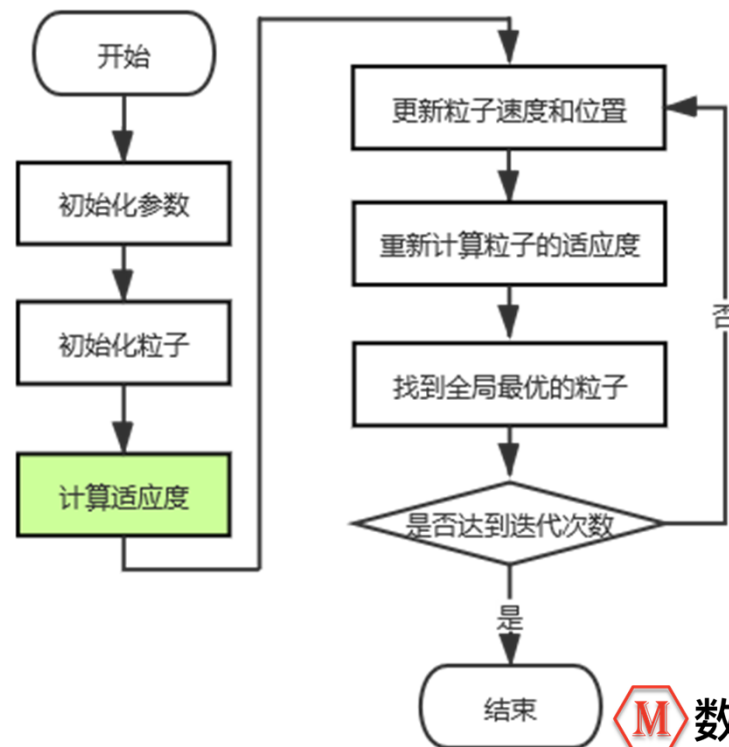
计算适应度

```
fit = zeros(n,1); % 初始化这n个粒子的适应度全为0
for i = 1:n % 循环整个粒子群, 计算每一个粒子的适应度
    fit(i) = Obj_fun1(x(i,:)); % 调用Obj_fun1函数来计算适应度
    (这里写成x(i,:)主要是为了和以后遇到的多元函数互通)
end
pbest = x; % 初始化这n个粒子迄今为止找到的最佳位置 (是一个n*narvs的向量)
ind = find(fit == max(fit), 1); % 找到适应度最大的那个粒子的下标
gbest = x(ind,:); % 定义所有粒子迄今为止找到的最佳位置 (是一个1*narvs的向量)
```

```
function y = Obj_fun1(x)
    y = 11*sin(x) + 7*cos(5*x);
end
```

注意:

- (1) 这里的适应度实际上就是我们的目标函数值。
- (2) 这里可以直接计算出pbest和gbest, 在后面将用于计算粒子的速度以更新粒子的位置。



循环体: 更新粒子速度和位置

```

for d = 1:K % 开始迭代, 一共迭代K次
    for i = 1:n % 依次更新第i个粒子的速度与位置
        % 更新第i个粒子的速度
        v(i,:) = w*v(i,:) + c1*rand(1)*(pbest(i,:) - x(i,:)) + c2*rand(1)*(gbest - x(i,:));
        % 如果粒子的速度超过了最大速度限制, 就对其进行调整
        for j = 1: narvs
            if v(i,j) < -vmax(j)
                v(i,j) = -vmax(j);
            elseif v(i,j) > vmax(j)
                v(i,j) = vmax(j);
            end
        end
        x(i,:) = x(i,:) + v(i,:); % 更新第i个粒子的位置
        % 如果粒子的位置超出了定义域, 就对其进行调整
        for j = 1: narvs
            if x(i,j) < x_lb(j)
                x(i,j) = x_lb(j);
            elseif x(i,j) > x_ub(j)
                x(i,j) = x_ub(j);
            end
        end
    end

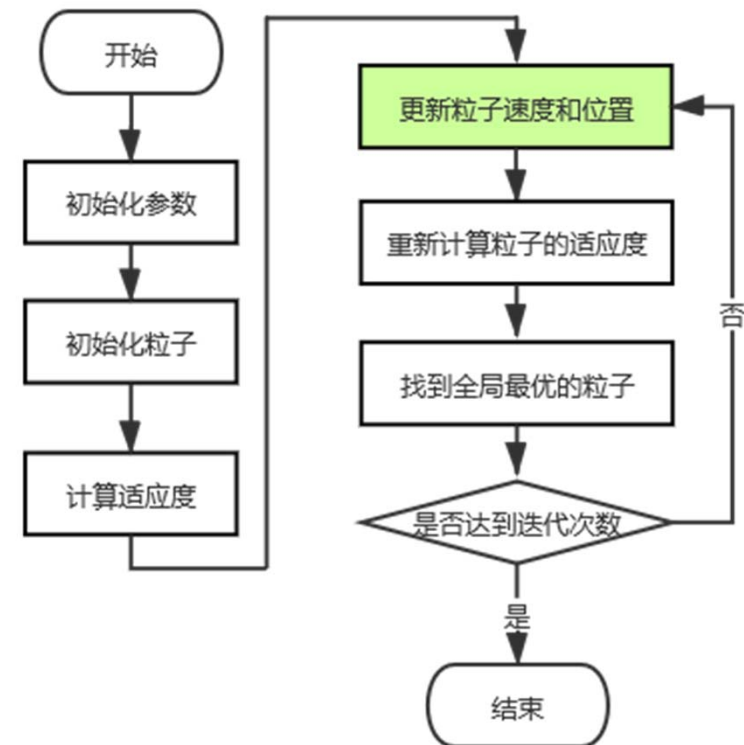
```

未完待续……

$$v_i^d = wv_i^{d-1} + c_1r_1(pbest_i^d - x_i^d) + c_2r_2(gbest^d - x_i^d)$$

其中 r_1, r_2 是 $[0, 1]$ 上的随机数

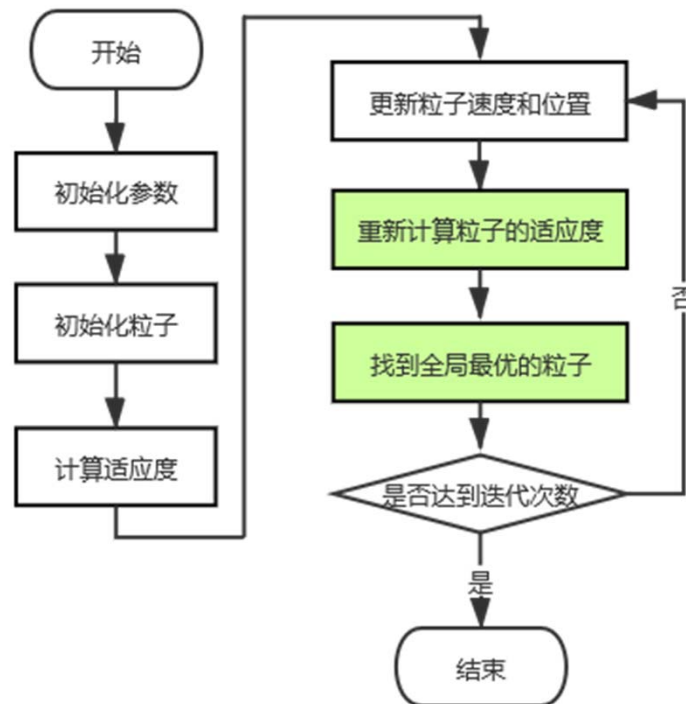
$$x_i^{d+1} = x_i^d + v_i^d$$



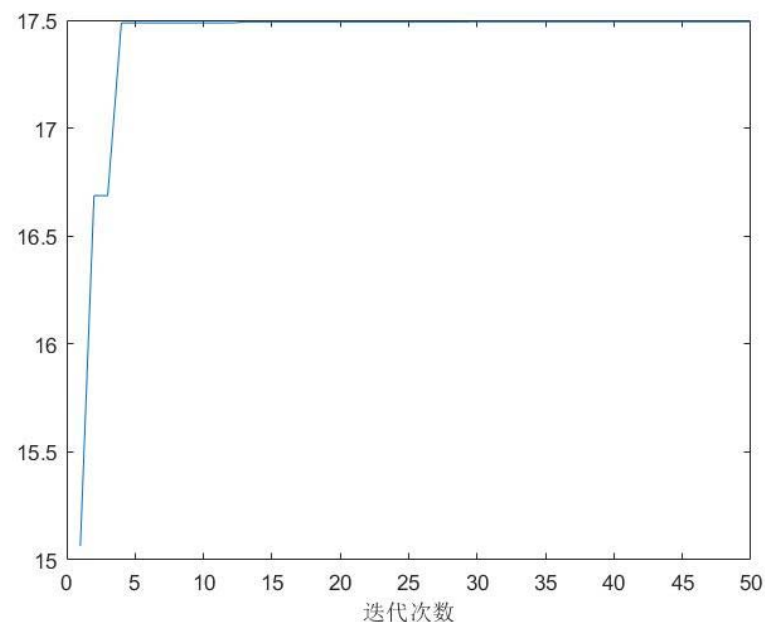
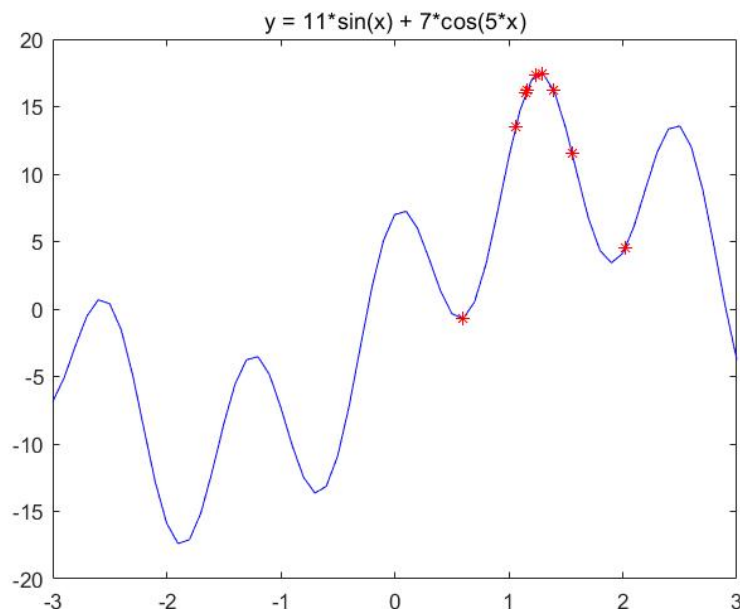
重新计算适应度并找到最优粒子

```

fit(i) = Obj_fun1(x(i,:)); % 重新计算第i个粒子的适应度
% 如果第i个粒子的适应度大于这个粒子迄今为止找到的最佳位置对应的适应度
if fit(i) > Obj_fun1(pbest(i,:))
    pbest(i,:) = x(i,:); % 那就更新第i个粒子迄今为止找到的最佳位置
end
% 如果第i个粒子的适应度大于所有的粒子迄今为止找到的最佳位置对应的适应度
if Obj_fun1(pbest(i,:)) > Obj_fun1(gbest)
    gbest = pbest(i,:); % 那就更新所有粒子迄今为止找到的最佳位置
end
end
end
end
    
```



粒子群算法运行的结果



```
n = 10; % 粒子数量  
c1 = 2; % 每个粒子的个体学习因子  
c2 = 2; % 每个粒子的社会学习因子  
w = 0.9; % 惯性权重  
K = 50; % 迭代的次数
```

最佳的位置是:
1.2748

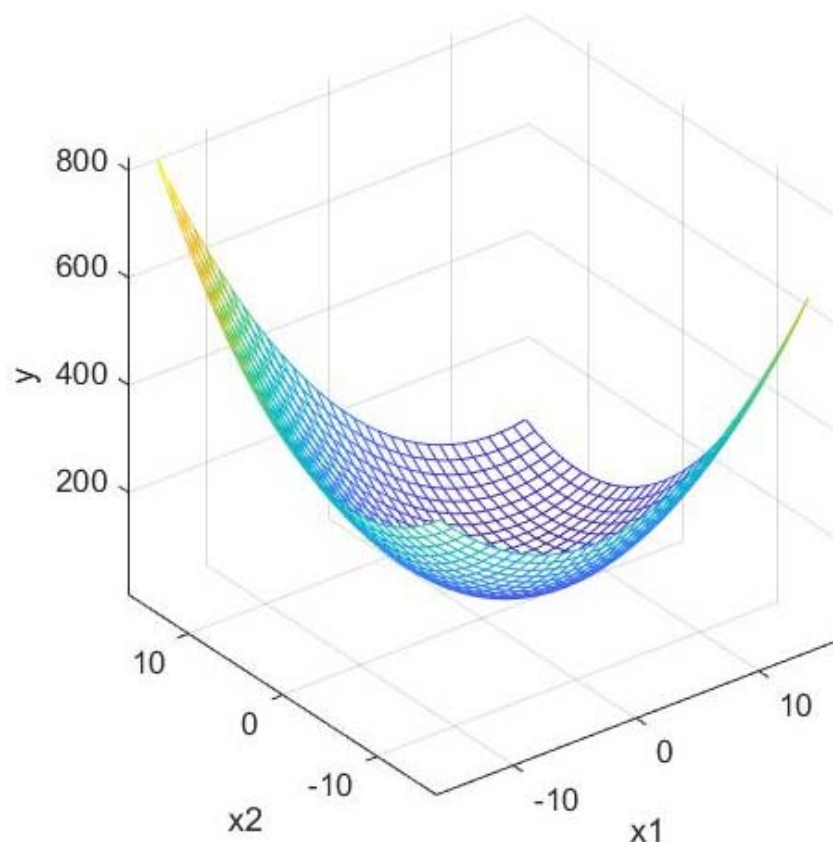
此时最优值是:
17.4928

注意: 每次运行的结果可能有差异哦~

例2: 求二元函数的最小值

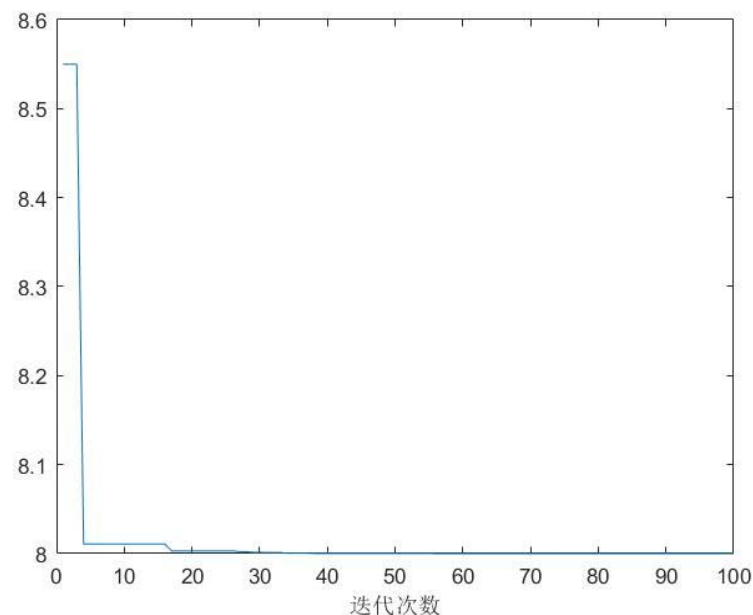
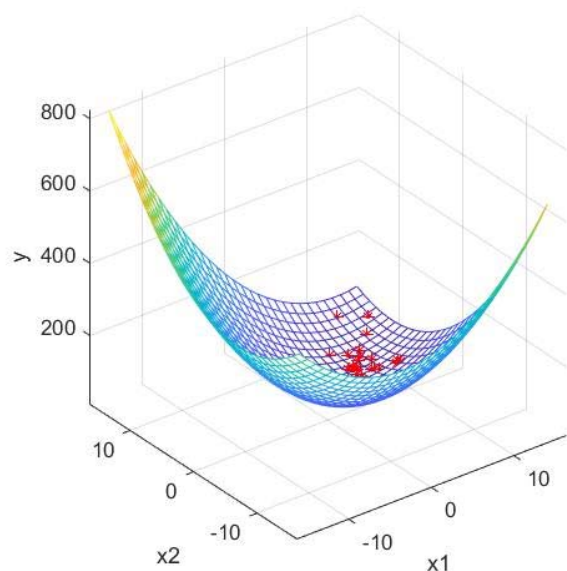
打开code2.m代码演示

求函数 $y = x_1^2 + x_2^2 - x_1x_2 - 10x_1 - 4x_2 + 60$ 在 $x_1, x_2 \in [-15, 15]$ 内的最小值



注意: 用粒子群算法求解函数最小值时, 粒子适应度的计算我们仍设置为目标函数值, 但是此时我们希望找到适应度最小的解。因此希望大家不要用我们中文的内涵去理解这里的“适应度” (中文的内涵就是越适应越好), 为了避免混淆你可以就直接用目标函数值来代替适应度的说法。

粒子群算法运行的结果



```
n = 30; % 粒子数量
narvs = 2; % 变量个数
c1 = 2; % 每个粒子的个体学习因子
c2 = 2; % 每个粒子的社会学习因子
w = 0.9; % 惯性权重
K = 100; % 迭代的次数
```

最佳的位置是:
8.0145 6.0065

此时最优值是:
8.0002

改进: 线性递减惯性权重

打开code3.m代码演示

$$v_i^d = w v_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)$$

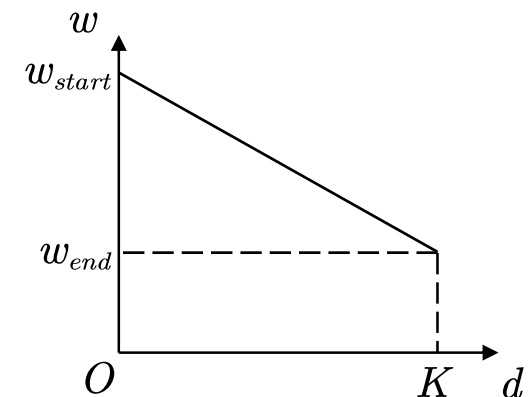
惯性权重 w 体现的是粒子继承先前的速度的能力, Shi,Y最先将惯性权重 w 引入到粒子群算法中, 并分析指出一个较大的惯性权值有利于全局搜索, 而一个较小的权值则更利于局部搜索。为了更好地平衡算法的全局搜索以及局部搜索能力, Shi,Y提出了线性递减惯性权重LDIW(Linear Decreasing Inertia Weight),公式如下:

$$w^d = w_{start} - (w_{start} - w_{end}) \times \frac{d}{K}$$

其中 d 是当前迭代的次数, K 是迭代总次数

w_{start} 一般取0.9, w_{end} 一般取0.4

与原来的相比, 现在惯性权重和迭代次数有关



参考论文: Shi, Y. and Eberhart, R.C. (1999) Empirical Study of Particle Swarm Optimization. Proceedings of the 1999 Congress on Evolutionary Computation, Washington DC, 6-9 July 1999, 1945-1950.

 数学建模学习交流

拓展: 非线性递减惯性权重

Baidu学术

粒子群非线性递减惯性权重

高级搜索 ▾

时间 ^

2020以来 (0)

2019以来 (56)

2018以来 (79)

年 - 年 确认

领域 ^

计算机科学与... (588)

控制科学与工程 (452)

电气工程 (438)

找到约21,600条相关结果

按相关性

一种非线性递减惯性权重策略的粒子群优化算法

目的 改进基本粒子群算法的一些缺点.基本粒子群算法是一种有效的寻找函数极值的演化计算方法,它简便易行,收敛速度快.但此算法也存在收敛精度不高,易陷入局部极值点的缺点.方法 对原有算法中的固定惯性权重进行改进.结果 提出一种非线性递减惯性权重策略...

李会荣,高岳林,李济民 - 《商洛学院学报》 - 被引量: 36

来源: 万方 / 知网 / 维普 / 爱学术 / 爱学术 ▾

收藏

引用

批量引用

$$w^d = w_{start} - (w_{start} - w_{end}) \times \left(\frac{d}{K}\right)^2$$

$$w^d = w_{start} - (w_{start} - w_{end}) \times \left[\frac{2d}{K} - \left(\frac{d}{K}\right)^2\right]$$

相关文献太多了, 不同学者可能得到的结论不同, 因为其求解的问题有差异

数学建模学习交流

三种递减惯性权重的图形

w_start = 0.9; w_end = 0.4;

K = 30; d = 1:K;

w1 = w_start-(w_start-w_end)*d/K;

w2 = w_start-(w_start-w_end)*(d/K).^2;

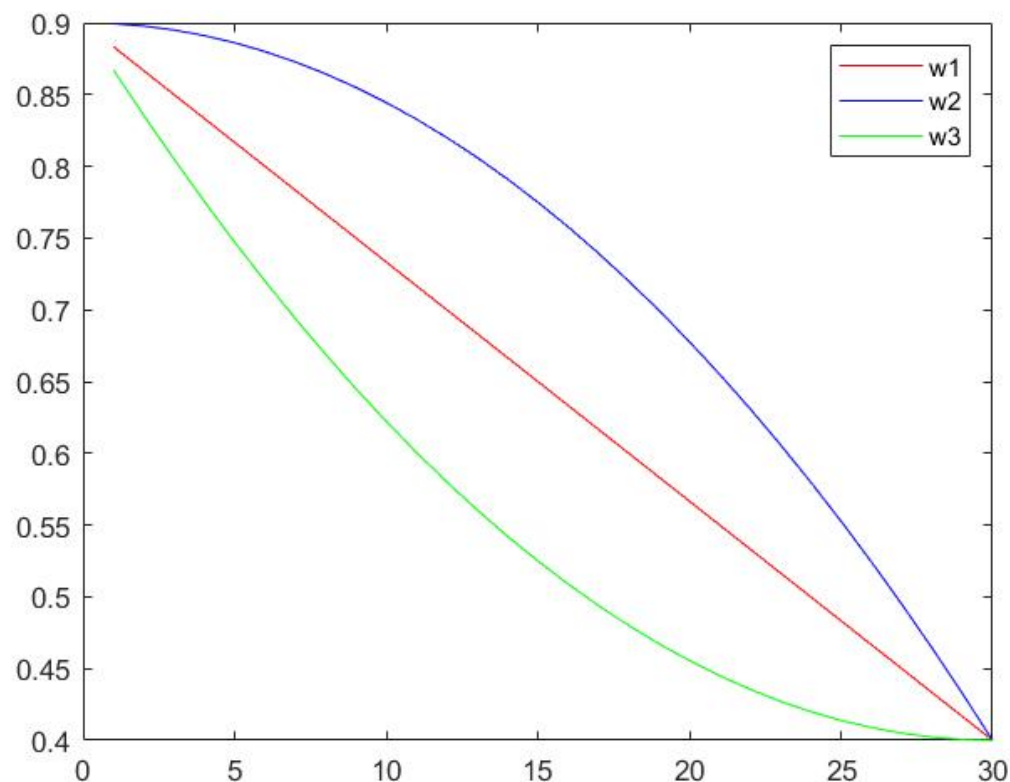
w3 = w_start-(w_start-w_end)*(2*d/K-(d/K).^2);

plot(d,w1,'r',d,w2,'b',d,w3,'g'); legend('w1','w2','w3')

$$w_1^d = w_{start} - (w_{start} - w_{end}) \times \frac{d}{K}$$

$$w_2^d = w_{start} - (w_{start} - w_{end}) \times \left(\frac{d}{K}\right)^2$$

$$w_3^d = w_{start} - (w_{start} - w_{end}) \times \left[\frac{2d}{K} - \left(\frac{d}{K}\right)^2\right]$$



数学建模学习交流

改进: 自适应惯性权重

打开code4.m代码演示

$$\mathbf{v}_i^d = \mathbf{w} \mathbf{v}_i^{d-1} + \mathbf{c}_1 \mathbf{r}_1 (\mathbf{pbest}_i^d - \mathbf{x}_i^d) + \mathbf{c}_2 \mathbf{r}_2 (\mathbf{gbest}^d - \mathbf{x}_i^d)$$

假设现在求最小值问题, 那么:

$$w_i^d = \begin{cases} w_{\min} + (w_{\max} - w_{\min}) \frac{f(x_i^d) - f_{\min}^d}{f_{\text{average}}^d - f_{\min}^d}, & f(x_i^d) \leq f_{\text{average}}^d \\ w_{\max} & , f(x_i^d) > f_{\text{average}}^d \end{cases}$$

其中:

- (1) w_{\min} 和 w_{\max} 是我们预先给定的最小惯性系数和最大惯性系数, 一般取0.4和0.9
- (2) $f_{\text{average}}^d = \sum_{i=1}^n f(x_i^d)/n$, 即第 d 次迭代时所有粒子的平均适应度
- (3) $f_{\min}^d = \min \{f(x_1^d), f(x_2^d), \dots, f(x_n^d)\}$, 即第 d 次迭代时所有粒子的最小适应度

与原来的相比, 现在惯性权重和迭代次数以及每个粒子适应度有关

 数学建模学习交流

改进: 自适应惯性权重

$$\mathbf{v}_i^d = \mathbf{w} \mathbf{v}_i^{d-1} + \mathbf{c}_1 \mathbf{r}_1 (\mathbf{pbest}_i^d - \mathbf{x}_i^d) + \mathbf{c}_2 \mathbf{r}_2 (\mathbf{gbest}^d - \mathbf{x}_i^d)$$

假设现在求**最小值问题**, 那么:

$$w_i^d = \begin{cases} w_{\min} + (w_{\max} - w_{\min}) \frac{f(x_i^d) - f_{\min}^d}{f_{\text{average}}^d - f_{\min}^d}, & f(x_i^d) \leq f_{\text{average}}^d \\ w_{\max} & , f(x_i^d) > f_{\text{average}}^d \end{cases}$$

一个较大的惯性权值有利于全局搜索
而一个较小的权值则更利于局部搜索

假设现在一共五个粒子ABCDE, 此时它们的适应度分别为1, 2, 3, 4, 5

取最大惯性权重为0.9, 最小惯性权重为0.4

那么, 这五个粒子的惯性权重应该为: 0.4, 0.65, 0.9, 0.9, 0.9

适应度越小, 说明距离最优解越近, 此时更需要局部搜索

适应度越大, 说明距离最优解越远, 此时更需要全局搜索

改进：自适应惯性权重

$$\mathbf{v}_i^d = \mathbf{w} \mathbf{v}_i^{d-1} + \mathbf{c}_1 \mathbf{r}_1 (\mathbf{pbest}_i^d - \mathbf{x}_i^d) + \mathbf{c}_2 \mathbf{r}_2 (\mathbf{gbest}^d - \mathbf{x}_i^d)$$


假设现在求**最大值问题**，那么：

$$w_i^d = \begin{cases} w_{\min} + (w_{\max} - w_{\min}) \frac{f_{\max}^d - f(x_i^d)}{f_{\max}^d - f_{\text{average}}^d}, & f(x_i^d) \geq f_{\text{average}}^d \\ w_{\max} & , f(x_i^d) < f_{\text{average}}^d \end{cases}$$

其中：

- (1) w_{\min} 和 w_{\max} 是我们预先给定的最小惯性系数和最大惯性系数，一般取0.4和0.9
- (2) $f_{\text{average}}^d = \sum_{i=1}^n f(x_i^d)/n$ ，即第 d 次迭代时所有粒子的平均适应度
- (3) $f_{\max}^d = \max\{f(x_1^d), f(x_2^d), \dots, f(x_n^d)\}$ ，即第 d 次迭代时所有粒子的最大适应度

与原来的相比，现在惯性权重和迭代次数以及每个粒子适应度有关

 数学建模学习交流

改进: 自适应惯性权重

$$\mathbf{v}_i^d = \mathbf{w}\mathbf{v}_i^{d-1} + \mathbf{c}_1\mathbf{r}_1(\mathbf{pbest}_i^d - \mathbf{x}_i^d) + \mathbf{c}_2\mathbf{r}_2(\mathbf{gbest}^d - \mathbf{x}_i^d)$$

假设现在求**最大值问题**, 那么:

$$w_i^d = \begin{cases} w_{\min} + (w_{\max} - w_{\min}) \frac{f_{\max}^d - f(x_i^d)}{f_{\max}^d - f_{\text{average}}^d}, & f(x_i^d) \geq f_{\text{average}}^d \\ w_{\max} & , f(x_i^d) < f_{\text{average}}^d \end{cases}$$

一个较大的惯性权值有利于全局搜索
而一个较小的权值则更利于局部搜索

假设现在一共五个粒子ABCDE, 此时它们的适应度分别为1, 2, 3, 4, 5

取最大惯性权重为0.9, 最小惯性权重为0.4

那么, 这五个粒子的惯性权重应该为: 0.9, 0.9, 0.9, 0.65, 0.4

适应度越小, 说明距离最优解越远, 此时更需要全局搜索

适应度越大, 说明距离最优解越近, 此时更需要局部搜索

改进: 随机惯性权重

最开始提出随机惯性权重的论文: Zhang L, Yu H, Hu S. A New Approach to Improve Particle Swarm Optimization[J]. lecture notes in computer science, 2003, 2723:134-139.

$$v_{id} = r_0 v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \quad (6)$$

where r_0 is a random number uniformly distributed in $[0,1]$, and the other parameters are same as before.

Our method can overcome two drawbacks of LDW. For one thing, decreasing the dependence of inertial weight on the maximum iteration that is difficultly predicted before experiments. Another is avoiding the lacks of local search ability at early of run and global search ability at the end of run.

**使用随机的惯性权重, 可以避免在迭代前期局部搜索能力的不足;
也可以避免在迭代后期全局搜索能力的不足。**

改进: 随机惯性权重

打开code5.m代码演示

$$v_i^d = w v_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)$$

参考文献: 基于随机惯性权重的简化粒子群优化算法[J]. 计算机应用研究, 2014, 031(002):361-363,391.

基于以上分析,提出随机惯性权重 ω 生成公式如下:

$$\omega = \mu_{\min} + (\mu_{\max} - \mu_{\min}) \times \text{rand}() + \sigma \times \text{randn}() \quad (3)$$

其中: μ_{\min} 是随机惯性权重的最小值; μ_{\max} 是随机惯性权重的最大值; $\text{rand}()$ 为 $[0, 1]$ 均匀分布随机数; 第三项中 $\text{randn}()$ 为正态分布的随机数; σ (方差) 用来度量随机变量权重 ω 与其数学期望 (即均值) 之间的偏离程度, 该项是为了控制取值中的权重误差, 使权重 ω 有利于向期望权重方向进化, 这样做的依据是正常情况下实验误差服从正态分布。

应该是标准差, 一般取0.2-0.5之间的一个数

其他关于惯性权重的论文

高级搜索

订阅

时间

2020以来 (0)

2019以来 (0)

2018以来 (0)

-

领域

计算机科学与... (3)

电气工程 (2)

控制科学与工程 (1)

+

核心

中国科技核心... (2)

北大核心期刊 (1)

SCI索引 (0)

+

获取方式

免费下载 (5)

找到28条相关结果

按相关性

粒子群优化算法中惯性权重综述

粒子群优化(particle swarm optimization,PSO)算法是基于鸟群觅食行为的一种新型的群体智能算法,而惯性权重是PSO算法中一个极其重要的参数,其值的选取直接关系粒子在寻优过程中的开发能力和探索能力。在介绍PSO算法的基本原理的基础上,分析惯性权重对粒...

周俊, 陈璟华, 刘国祥, ... - 《广东电力》 - 被引量: 21

来源: 知网 / 万方 / 维普 / 爱学术 / 爱学术

收藏

引用

批量引用

粒子群优化算法中惯性权重的研究进展

粒子群优化算法是根据鸟群觅食过程中的迁徙和群集模型而提出的用于解决优化问题的一类新兴的随机优化算法.惯性权重是粒子群算法中非常重要的参数,可以用来控制算法的开发和探索能力.简单介绍了标准粒子群优化算法的基本原理,全面综述了现有文献中对惯性...

田雨波, 朱人杰, 薛权祥 - 《计算机工程与应用》 - 被引量: 77 - 2008年

来源: 万方 / 知网 / 维普 / 《计算机工程与应用...》 / 《计算机工程与应用...》

收藏

引用

批量引用

数学建模学习交流

关注微信公众号"数学建模学习交流"获取更多优质资料

39 / 59

改进: 压缩(收缩)因子法

$$\mathbf{v}_i^d = w\mathbf{v}_i^{d-1} + c_1 r_1 (\mathbf{pbest}_i^d - \mathbf{x}_i^d) + c_2 r_2 (\mathbf{gbest}^d - \mathbf{x}_i^d)$$

个体学习因子 c_1 和社会(群体)学习因子 c_2 决定了粒子本身经验信息和其他粒子的经验信息对粒子运行轨迹的影响, 其反映了粒子群之间的信息交流。设置 c_1 较大的值, 会使粒子过多地在自身的局部范围内搜索, 而较大的 c_2 的值, 则又会促使粒子过早收敛到局部最优值。

为了有效地控制粒子的飞行速度, 使算法达到全局搜索与局部搜索两者间的有效平衡, **Clerc构造了引入收缩因子的PSO模型, 采用了压缩因子, 这种调整方法通过合适选取参数, 可确保PSO算法的收敛性, 并可取消对速度的边界限制。**

参考文献: M. Clerc. The swarm and queen: towards a deterministic and adaptive particle swarm optimization. Proc. Congress on Evolutionary Computation, Washington, DC, Piscataway, NJ:IEEE Service Center (1999) 1951–1957

改进: 压缩因子法

打开code6.m代码演示

Recent work done by Clerc (1999) indicates that use of a *constriction factor* may be necessary to insure convergence of the particle swarm algorithm. A detailed discussion of the constriction factor is beyond the scope of this paper, but a simplified method of incorporating it appears in equation (3), where K is a function of c_1 and c_2 as reflected in equation (4).

$$v_{id} = K * [v_{id} + c_1 * rand() * (p_{id} - x_{id}) + c_2 * Rand() * (p_{gd} - x_{id})] \quad (3)$$

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|}, \text{ where } \varphi = c_1 + c_2, \varphi > 4 \quad (4)$$

参考文献:

Eberhart R C . Comparing inertia weights and constriction factors in optimization[C]//
Proceedings of the 2000 IEEE Congress on
Evolutionary Computation, La Jolla, CA. IEEE, 2000.

压缩因子法中应用较多的个体学习因子 c_1 和社会学习因子 c_2 均取2.05,
用我们自己的符号可以表示为:

$$c_1 = c_2 = 2.05, C = c_1 + c_2 = 4.1, \text{ 收缩因子 } \Phi = \frac{2}{|(2 - C - \sqrt{C^2 - 4C})|};$$

惯性权重 $w = 0.9$, 速度更新公式改为:

$$v_i^d = \Phi \times [wv_i^{d-1} + c_1 r_1 (pbest_i^d - x_i^d) + c_2 r_2 (gbest^d - x_i^d)]$$

改进: 非对称学习因子

4 学习因子线性变化的粒子群算法

在经典 PSO 算法中, 由于在寻优后期粒子缺乏多样性, 易过早收敛于局部极值^[7], 因此通过调节学习因子, 在搜索初期使粒子进行大范围搜索, 以期获得具有更好多样性的高质量粒子, 尽可能摆脱局部极值的干扰。

学习因子 $c1$ 和 $c2$ 决定粒子个体经验信息和其他粒子经验信息对寻优轨迹的影响, 反映了粒子之间的信息交换。设置较大的 $c1$ 值, 会使粒子过多的在局部搜索; 反之, 较大的 $c2$ 值会使粒子过早收敛到局部最优值。因此, 在算法搜索初期采用较大的 $c1$ 值和较小的 $c2$ 值, 使粒子尽量发散到搜索空间即强调“个体独立意识”, 而较少受到种群内其他粒子即“社会意识部分”的影响, 以增加群内粒子的多样性。随着迭代次数的增加, 使 $c1$ 线性递减, $c2$ 线性递增, 从而加强了粒子向全局最优点的收敛能力:

这里的局部搜索指的是粒子会过多的在自身的局部范围内进行搜索, 从全局来看实际上增大了搜索范围。(可以想象成每个粒子各干各的事情, 团队意识比较弱)

综合以上分析, 基于 $c1=2.5-0.5$, $c2=1.0-2.25$ 的非对称学习因子调节策略与文献[5]的对称调节以及学习因子固定取值的方法相比, 均有较为明显的改善, 表现出较好的全局寻优能力。

减号应该改为加号

$$c1 = c_{1i} - k \times (c_{1f} - c_{1i}) / k_{\max} \quad (10)$$

$$c2 = c_{2i} + k \times (c_{2f} - c_{2i}) / k_{\max} \quad (11)$$

其中, k 为当前迭代次数; k_{\max} 是最大迭代数; c_{1i} 、 c_{2i} 分别为 $c1$ 、 $c2$ 的初始值; c_{1f} 、 c_{2f} 分别为 $c1$ 、 $c2$ 的最终值。

根据上述收敛性分析, 在迭代过程中 w 值从 1 线性递减到 0.4, 参数 $c1$ 和 $c2$ 在满足 $c=c1+c2 \leq 4$ 的条件下, 有 3 种取值情况:

(1) $c1$ 和 $c2$ 是常数且 $c1=c2$ 。表示“个体”与“群体”对粒子搜索过程的影响力相同。

(2) $c1$ 和 $c2$ 是对称的线性变化关系。即 $c_{1i}=c_{2f}$ 且 $c_{2i}=c_{1f}$, 表示“个体”与“群体”对搜索过程起完全互补的作用, 这是经典 PSO 的思想。

(3) $c1$ 和 $c2$ 是非对称的线性变化关系。即 $c_{1i} \neq c_{2f}$ 且 $c_{2i} \neq c_{1f}$, 表示“个体”与“群体”对搜索过程具有不同程度的影响, 这也是本文的改进 PSO 算法。

参考文献: 毛开富, 包广清, 徐驰. 基于非对称学习因子调节的粒子群优化算法[J]. 计算机工程, 2010(19):188-190.

改进: 非对称学习因子

打开code7.m代码演示

$$c_1^{ini} = 2.5, c_1^{fin} = 0.5, c_2^{ini} = 1, c_2^{fin} = 2.25$$

$$c_1^d = c_1^{ini} + (c_1^{fin} - c_1^{ini}) \times \frac{d}{K}; c_2^d = c_2^{ini} + (c_2^{fin} - c_2^{ini}) \times \frac{d}{K}$$

其中 d 是当前迭代的次数, K 是迭代总次数

```
c1_ini = 2.5; % 个体学习因子的初始值
c1_fin = 0.5; % 个体学习因子的最终值
c2_ini = 1; % 社会学习因子的初始值
c2_fin = 2.25; % 社会学习因子的最终值
```

```
.....
```

```
for d = 1:K % 开始迭代, 一共迭代K次
```

```
    c1 = c1_ini + (c1_fin - c1_ini)*d/K;
```

```
    c2 = c2_ini + (c2_fin - c2_ini)*d/K;
```

```
    for i = 1:n % 依次更新第i个粒子的速度与位置
```

```
        % 更新第i个粒子的速度
```

```
        v(i,:) = w*v(i,:) + c1*rand(1)*(pbest(i,:) - x(i,:)) + c2*rand(1)*(gbest - x(i,:));
```

```
.....
```

优化问题的测试函数

打开code8.m代码演示


当你提出了一种新的优化算法后, 你需要和别人之前提出的算法来进行PK, 看你的算法有没有提高, 下表给出了四种常见的测试函数:

函数名	函数表达式	维数	取值范围	理论极值	误差目标
Sphere	$f(x) = \sum_{i=1}^n x_i^2$	30	$[-100, 100]^n$	0	10^{-2}
Rosenbrock	$f(x) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	30	$[-30, 30]^n$	0	10^2
Rastrigin	$f(x) = \sum_{i=1}^n (x_i^2 - 10 \cos(2\pi x_i) + 10)$	30	$[-5.12, 5.12]^n$	0	10^2
Griewank	$f(x) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30	$[-600, 600]^n$	0	10^{-1}

- 维数: 自变量x的个数, 也是上面表达式中n的大小
- 取值范围: 每个x对应的变化范围
- 理论极值: 这个函数理论上的最小值
- 误差目标: 只要我们求出来的最小值小于这个目标值就能被接受

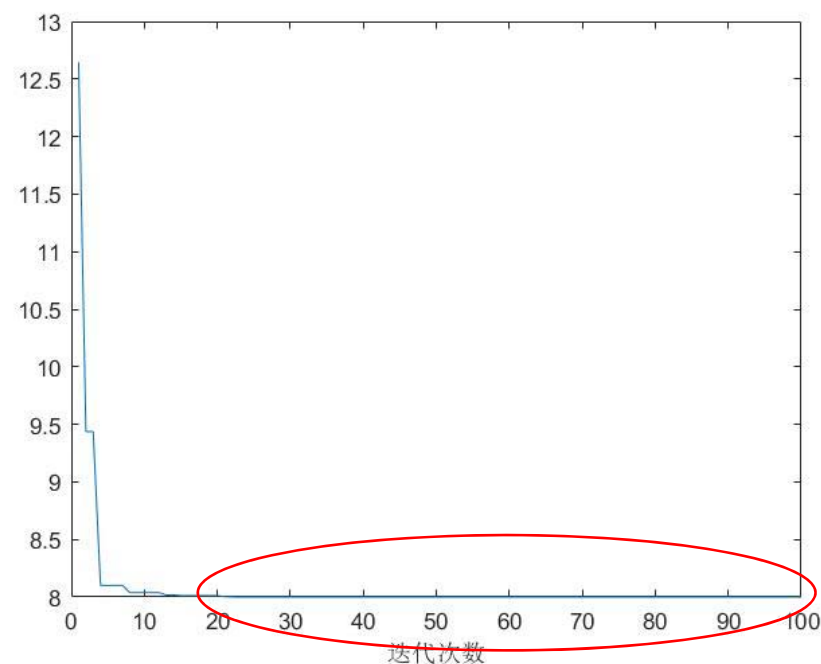
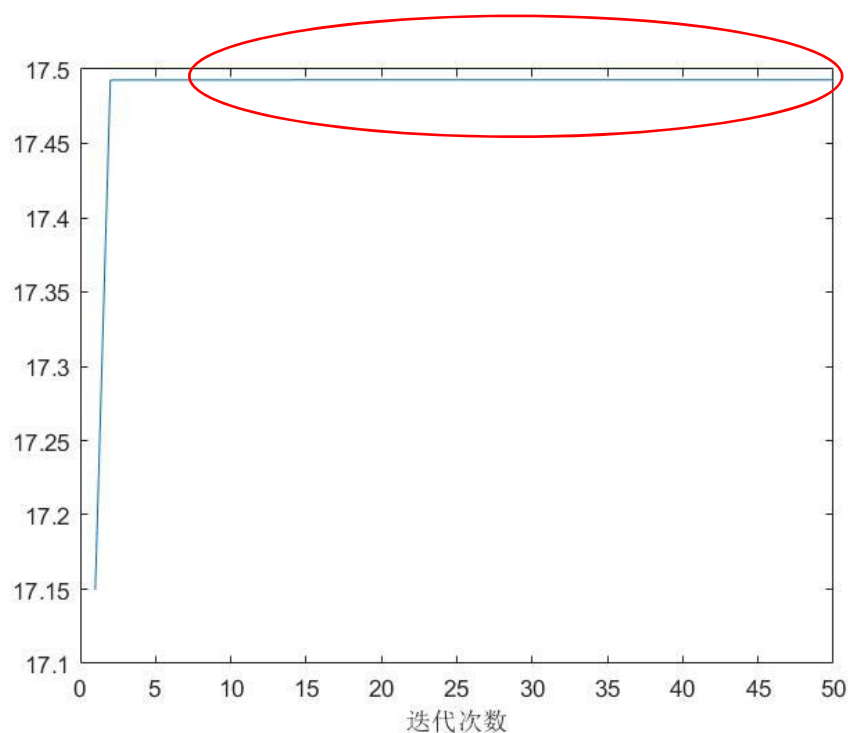
下面我们就来用粒子群算法求解这四个测试函数

张玮, 王华奎. 粒子群算法稳定性的参数选择策略分析[J]. 系统仿真学报, 2009, 21(014):4339-4344.

 数学建模学习交流

改进: 自动退出迭代循环

当粒子已经找到最佳位置后, 再增加迭代次数只会浪费计算时间, 那么我们能否设计一个策略, 能够自动退出迭代呢?



改进: 自动退出迭代循环

打开code9.m代码演示

当粒子已经找到最佳位置后, 再增加迭代次数只会浪费计算时间, 那么我们能否设计一个策略, 能够自动退出迭代呢?

- (1) 初始化最大迭代次数、计数器以及最大计数值 (例如分别取100, 0, 20)
- (2) 定义“函数变化量容忍度”, 一般取非常小的正数, 例如 10^{-6} ;
- (3) 在迭代的过程中, 每次计算出来最佳适应度后, 都计算该适应度和上一次迭代时最佳适应度的变化量(取绝对值);
- (4) 判断这个变化量和“函数变化量容忍度”的相对大小, 如果前者小, 则计数器加1; 否则计数器清0;
- (5) 不断重复这个过程, 有以下两种可能:
 - ① 此时还没有超过最大迭代次数, 计数器的值超过了最大计数值, 那么我们就跳出迭代循环, 搜索结束。
 - ② 此时已经达到了最大迭代次数, 那么直接跳出循环, 搜索结束。

深入研究粒子群算法

前面我们学习的是最基础的粒子群

粒子群算法的发展^[10]始于 1995 年 Eberhart 和 Kennedy 提出的基本粒子群算法。其中基本 PSO 的参数是固定的,在对某些函数优化上精度较差。

后来 Shi 等^[11]提出了惯性因子 w 线性递减的改进算法,使算法在搜索初期具有较大搜索能力,而在后期又能够得到较精确的结果,此改进大大提高了基本 PSO 算法的性能。2001 年 Shi 又提出了自适应模糊调节 w 的 PSO,在对单峰函数的处理中取得了良好的效果。

van den Bergh^[12]通过使粒子群中最佳粒子始终处于运动状态,得到保证收敛到局部最优的改进算法,但其性能并不佳。

Kennedy 等^[13]研究粒子群的拓扑结构,分析粒子间的信息流,提出了一系列的拓扑结构。

Angeline^[14]将选择算子引入到 PSO 中,选择每次迭代后的较好的粒子并复制到下一代,以保证每次迭代的粒子群都具有较好的性能。


Higashi 等^[15]分别提出了自己的变异 PSO 算法,基本思路均是希望通过引入变异算子跳出局部极值点的吸引,从而提高算法的全局搜索能力,得到较高的搜索成功率。

Baskar 等^[16]各自提出了自己的协同 PSO 算法,通过使用多群粒子分别优化问题的不同维、多群粒子协同优化等办法来对基本算法进行改进尝试。

Al-Kazemi^[17]所提出的 Multi-Phase PSO 在粒子群中随机选取部分个体向全局最优飞,而其他个体向反方向飞,以扩大搜索空间。

除以上的混合算法之外,还出现了量子 PSO、模拟退火 PSO、耗散 PSO、自适应 PSO 等混合改进算法,也有采取 PSO 与基于梯度的优化方法相结合的办法等。

纪震等《粒子群算法及应用》科学出版社, 2009, P13-14

 数学建模学习交流

Matlab自带的粒子群函数

打开code10.m代码演示

particleswarm

Particle swarm optimization

Syntax

```
x = particleswarm(fun,nvars)
x = particleswarm(fun,nvars,lb,ub)
x = particleswarm(fun,nvars,lb,ub,options)
x = particleswarm(problem)
[x,fval,exitflag,output] = particleswarm(____)
```

Matlab自带的这个函数优化的特别好, 运行速度快且找到的解也比较精准, 但是内部的实现比较复杂, 后面我们会简单介绍这个函数的实现思路。


(代码如何跑得快: 少写循环和判断语句, 多基于矩阵运算来进行操作)

下面我们用Matlab这个内置粒子群函数为大家演示如何求 *Rosenbrock* 函数的最小值。

注意:

- (1) 这个函数在R2014b版本后推出, 之前的老版本Matlab中不存在。
- (2) 这个函数是求最小值的, 如果目标函数是求最大值则需要添加负号从而转换为求最小值。

参考资料: Matlab自带的粒子群优化函数particleswarm.pdf

 数学建模学习交流

Matlab粒子群函数的细节讲解

钱锋《粒子群算法及其工业应用》科学出版社, 2013, P34

Kennedy 等在观察鸟群觅食的过程中注意到, 通常飞鸟并不一定看到鸟群中其他所有飞鸟的位置和方向, 往往只是看到相邻的飞鸟的位置和方向。因此他在研究粒子群算法时, 同时开发了两种模式: 全局模式 (gbest) 和邻域模式 (lbest)^[17,18]。

全局模式是指粒子群在搜索过程中将所有其他粒子都视为邻域粒子, 它可以看做是邻域模式的极端情况, 基本粒子群算法即属于全局模式。其优点是粒子邻域个体多, 粒子群内的信息交流速度快, 使得粒子群算法具有较快的收敛速度, 但在一定程度上会降低粒子多样性, 易陷入局部最优。

邻域模式是指粒子群在搜索过程中只将其周围部分粒子视为邻域粒子, 这种模式使得粒子群可以被分割成多个不同的子群体, 有利于在多个区域进行搜索, 避免算法陷入局部最优。

搜索初期使用邻域模式较好, 后期使用全局模式

Matlab中particleswarm函数采用的是自适应的邻域模式

该函数主要参考的两篇文章: (该信息可在Matlab官网中查询)

[1] Mezura-Montes, E., and C. A. Coello Coello. "Constraint-handling in nature-inspired numerical optimization: Past, present and future." Swarm and Evolutionary Computation. 2011, pp. 173–194.

[2] Pedersen, M. E. "Good Parameters for Particle Swarm Optimization." Luxembourg: Hvass Laboratories, 2010.

预设参数的选取

(1) 粒子个数 **SwarmSize**

默认设置为: $\min\{100, 10 \times \text{nvars}\}$, nvars是变量个数

(2) 惯性权重 **InertiaRange**

默认设置的范围为: $[0.1, 1.1]$, 注意, 在迭代过程中惯性权重会采取自适应措施, 随着迭代过程不断调整。

(3) 个体学习因子 **SelfAdjustmentWeight**

默认设置为: 1.49 (和压缩因子的系数几乎相同)

(4) 社会学习因子 **SocialAdjustmentWeight**

默认设置为: 1.49 (和压缩因子的系数几乎相同)

(5) 邻域内粒子的比例 **MinNeighborsFraction**

默认设置为: 0.25, 由于采取的是邻域模式, 因此定义了一个“邻域最少粒子数目”: $\text{minNeighborhoodSize} = \max\{2, (\text{粒子数目} \times \text{邻域内粒子的比例}) \text{的整数部分}\}$, 在迭代开始后, 每个粒子会有一个邻域, 初始时邻域内的粒子个数(记为Q)就等于“邻域最少粒子数目”, 后续邻域内的粒子个数Q会自适应调整。

变量初始化和适应度的计算

(1) 速度初始化: 和我们之前的类似, 只不过最大速度就是上界和下界的差额

$v_{\max} = ub - lb; \quad v = -v_{\max} + 2 * v_{\max} .* \text{rand}(n, \text{narvs});$

(2) 位置初始化: 和我们之前的类似

会将每个粒子的位置均匀分布在上下界约束内

(3) 计算每个粒子的适应度

适应度仍设置为我们要优化的目标函数, 由于该函数求解的是最小值问题, 因此, 最优解应为适应度最小即目标函数越小的解。

(4) 初始化个体最优位置

和我们自己写的代码一样, 因为还没有开始循环, 因此这里的个体最优位置就是我们初始化得到的位置。

(5) 初始化所有粒子的最优位置

因为每个粒子的适应度我们都已经求出来了, 所以我们只需要找到适应度最低的那个粒子, 并记其位置为所有粒子的最优位置。

更新粒子的速度和位置

在每次迭代中, 我们要分别更新每一个粒子的信息。例如: 对于现在要更新的粒子 i , 我们要进行以下几个操作:

(1) 随机生成粒子 i 的邻域, 邻域内一共 Q 个粒子(包含粒子 i 本身), 并找到这 Q 个粒子中位置最佳的那个粒子, 此时它的目标函数值最小, 记其位置为 $lbest$;

(2) 更新粒子 i 的速度, 公式为: $v = w*v + c1*u1*(pbest-x) + c2*u2*(lbest-x)$;
(注: 这里省略了一些上下标, 大家应该能理解), 这个速度公式和基本粒子群算法最大的不同在于: 这里的群体信息交流部分使用的是邻域内的最优位置, 而不是整个粒子群的最优位置;

(3) 更新粒子 i 的位置, 公式为: $x = x + v$, 这个和基本粒子群算法一样;

(4) 修正位置和速度: 如果粒子 i 的位置超过了约束, 就将其位置修改到边界处; 另外如果这个粒子的位置在边界处, 我们还需要查看其速度是否超过了最大速度, 如果超过的话将这个速度变为0。(注意, 如果是多元函数的话可能只有某个分量超过了约束, 我们的修改只需要针对这个分量即可)

(5) 计算粒子 i 的适应度, 如果小于其历史最佳的适应度, 就更新粒子 i 的历史最佳位置为现在的位置; 另外还需要判断粒子 i 的适应度是否要小于所有粒子迄今为止找到的最小适应度, 如果小的话需要更新所有的粒子的最佳位置为粒子 i 的位置。

自适应调整参数

[1] Iadevaia, "Identification of optimal drug combinations targeting cellular networks: integrating phospho-proteomics and computational network analysis." Cancer Res. 70, 6704-6714 (2010).
[2] Liu, . "Parameter estimation in dynamic biochemical systems based on adaptive Particle Swarm Optimization." Information, Communications and Signal Processing, 2009. ICICS 2009.

假设在第 d 次迭代过程中, 所有的粒子的信息都已经更新好了, 那么在开始下一次的迭代之前, 需要更新模型中的参数, 这里就体现了自适应过程。

规则如下: 记此时所有粒子的最小适应度为 a , 上一次迭代完成后所有粒子的最小适应度为 b 。比较 a 和 b 的相对大小, 如果 $a < b$, 则记 $\text{flag} = 1$; 否则记 $\text{flag} = 0$ 。

如果: $\text{flag} = 0$, 那么做下面的操作:

(1) 更新 $c = c + 1$; 这里的 c 表示“停滞次数计数器”, 在开始迭代前就初始为0

(2) 更新 $Q = \min\{Q + \min\text{NeighborhoodSize}, \text{SwarmSize}\}$; Q : 邻域内的粒子个数;

$\min\text{NeighborhoodSize}$: 邻域最少粒子数目; SwarmSize : 粒子的总数

如果: $\text{flag} = 1$, 那么做下面的操作:

(1) 更新 $Q = \min\text{NeighborhoodSize}$

(2) 更新 $c = \max\{c - 1, 0\}$

(3) 判断 c 的大小, 如果 $c < 2$, 则更新 $w = 2 * w$; 如果 $c > 5$, 则更新 $w = w / 2$; 这里的 w 是惯性权重, 如果计算的结果超过了惯性权重的上界或低于下界都需要将其进行调整到边界处。

自适应体现在: 如果适应度开始停滞时, 粒子群搜索会从邻域模式向全局模式转换, 一旦适应度开始下降, 则又恢复到邻域模式, 以免陷入局部最优。当适应度的停滞次数足够大时, 惯性系数开始逐渐变小, 从而利于局部搜索。



自动退出迭代循环

Matlab自带的粒子群函数可以设置几种自动退出迭代循环的方法:

particleswarm iterates until it reaches a stopping criterion.

Stopping Option	Stopping Test	Exit Flag
MaxStallIterations and FunctionTolerance	Relative change in the best objective function value g over the last <code>MaxStallIterations</code> iterations is less than <code>FunctionTolerance</code> .	1
MaxIterations	Number of iterations reaches <code>MaxIterations</code> .	0
OutputFcn or PlotFcn	OutputFcn or PlotFcn can halt the iterations.	-1
ObjectiveLimit	Best objective function value g is less than or equal to <code>ObjectiveLimit</code> .	-3
MaxStallTime	Best objective function value g did not change in the last <code>MaxStallTime</code> seconds.	-4
MaxTime	Function run time exceeds <code>MaxTime</code> seconds.	-5

默认20

默认1e-6

默认
200*nvars

默认-inf

默认+inf

默认+inf

按照默认值的设置来解释这句话的意思:
从第21次迭代完成后开始, 以后每次迭代完成后都要计算一个变化量: $\text{funChange} = \text{abs}(\text{距离现在19期前的最优值} - \text{当前的最优值}) / \max\{1, \text{abs}(\text{当前的最优值})\}$;
如果变化量 $\text{funChange} < 1e-6$, 就退出迭代。
(这个超级容易弄错, 它和我们前面自己写的改进方案code9不同, 我也是对着源代码调试了好久才明白, 😊(/// ∩ ω ∩))

图形上面有停止迭代的按钮

If particleswarm stops with exit flag 1, it optionally calls a hybrid function after it exits.

如果是第一种方式退出迭代的话, 我们可以将粒子群算法得到的解作为初始值, 继续调用其他的函数来进行混合求解, 例如我们熟悉的fmincon函数 (我测试发现以flag0退出好像也可以调用其他函数混合求解)

修改函数的参数

函数中可以调整的参数 (不是完全的, 但足够了)	options中的设置
绘制最佳的函数值随迭代次数的变化图	'PlotFcn','pswplotbestf'
展示函数的迭代过程	'Display','iter'
修改粒子数量, 默认的是 $\min(100, 10 \times \text{nvars})$	'SwarmSize',50
粒子群算法结束后调用其他函数进行后续求解	'HybridFcn',@fmincon
惯性权重的变化范围, 默认的是0.1-1.1	'InertiaRange',[0.2 1.2]
个体学习因子, 默认的是1.49 (压缩因子)	'SelfAdjustmentWeight',2
社会学习因子, 默认的是1.49 (压缩因子)	'SocialAdjustmentWeight',2
最大的迭代次数, 默认的是 $200 \times \text{nvars}$	'MaxIterations',10000
邻域内粒子的比例, 默认的是0.25	'MinNeighborsFraction',0.2
函数容忍度, 默认 $1e-6$, 用于控制自动退出迭代的参数	'FunctionTolerance', $1e-8$
最大停滞迭代数MaxStallIterations, 默认20	'MaxStallIterations',50

参考: Matlab官网关于粒子群算法函数的说明

函数参数修改的建议

看函数运行完成后, Matlab输出的停止搜索的原因:

可能的原因1:

Optimization ended: relative change in the objective value over the last OPTIONS.MaxStallIterations iterations is less than OPTIONS.FunctionTolerance.

解决方法: 将'FunctionTolerance'改小, 'MaxStallIterations'增加

可能的原因2:

Optimization ended: number of iterations exceeded OPTIONS.MaxIterations.

解决方法: 将'MaxIterations'增加

如果不考虑计算时间, 那么我们可以同时修改上面的设置。

'FunctionTolerance',1e-12,'MaxStallIterations',100,'MaxIterations',100000

fval =

目标函数: Rosenbrock函数

8.7779e-10

当然我们也可以在粒子群结束后调用其他函数进行混合求解, 大家可以自己测试。

粒子群算法的后续讨论

(1) 粒子群算法可以解决非线性约束问题吗?

X. Hu, and R. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002), Orlando, USA

There are two modifications compared to the original PSO.

1. During initialization, all the particles are repeatedly initialized until they satisfy all the constraints.
2. When calculating the *pBest* and *gBest* values, only those positions in feasible space are counted.

Parsopoulos K E , Vrahatis M N . Particle Swarm Optimization Method for Constrained Optimization Problems[M]// Intelligent Technologies-Theory and Applications: New Trends in Intelligent Technologies. CiteSeer, 2002.

Abstract. The performance of the Particle Swarm Optimization method in coping with Constrained Optimization problems is investigated in this contribution. In the adopted approach a non-stationary multi-stage assignment penalty function is incorporated, and several experiments are performed on well-known and widely used benchmark problems. The obtained results are reported and compared with those obtained through different evolutionary algorithms, such as Evolution Strategies and Genetic Algorithms. Conclusions are derived and directions of future research are exposed.

在适应度函数上构造惩罚项, 对违反约束的情况进行惩罚,
将有约束的优化问题转化为无约束的优化问题



数学建模学习交流

粒子群算法的后续讨论

(2) 粒子群算法可以解决组合优化问题吗?

优化问题可分为两大类: 一类是具有连续型的变量, 比如我们之前探究的求函数最值的问题, 称为连续优化问题; 另一类是具有离散型的变量, 例如0-1规划问题、TSP旅行商问题, 称为组合优化问题。

组合优化问题就是在给定的约束条件下, 求出使目标函数极小(或极大)的变量组合问题。典型的组合优化问题有旅行商问题 (Traveling Salesman Problem—TSP)、加工调度问题、0-1 背包问题、装箱问题、聚类问题、指派问题等。它们均属于 NP 难问题, 它们具有共同的特点: 假设问题的规模为 n , 问题空间可以归结为由 n 个自然数的全排列组成的离散集合。当问题规模较大时, 没有合适的算法求精确解。求解这类 NP 问题最好的算法是智能优化算法 (如遗传算法、模拟退火、蚁群优化等)。智能优化算法受物理现象或仿生学机理等的启发产生。该类算法一般采用概率或随机搜索策略, 能够在可行时间内以较大概率获得问题的最优解或近似最优解, 从而保证算法在解的质量与计算费用之间获得较好的平衡。鉴于以上优点, 该类算法已经成为 NP 问题最常用的求解方法之一。

由于粒子群算法最初被提出是用于连续空间的优化, 从核心运动公式上看, 定义的都是连续的变量。组合优化问题有其特殊性, 原来的运动公式已经不能适用于求解某些组合优化问题。

解决思路:

- (1) 对原来粒子群算法的运动公式和运算符号进行重新定义;
- (2) 利用其他的智能算法和粒子群算法结合, 进行混合求解。

这类问题后面有更好的办法解决: 模拟退火

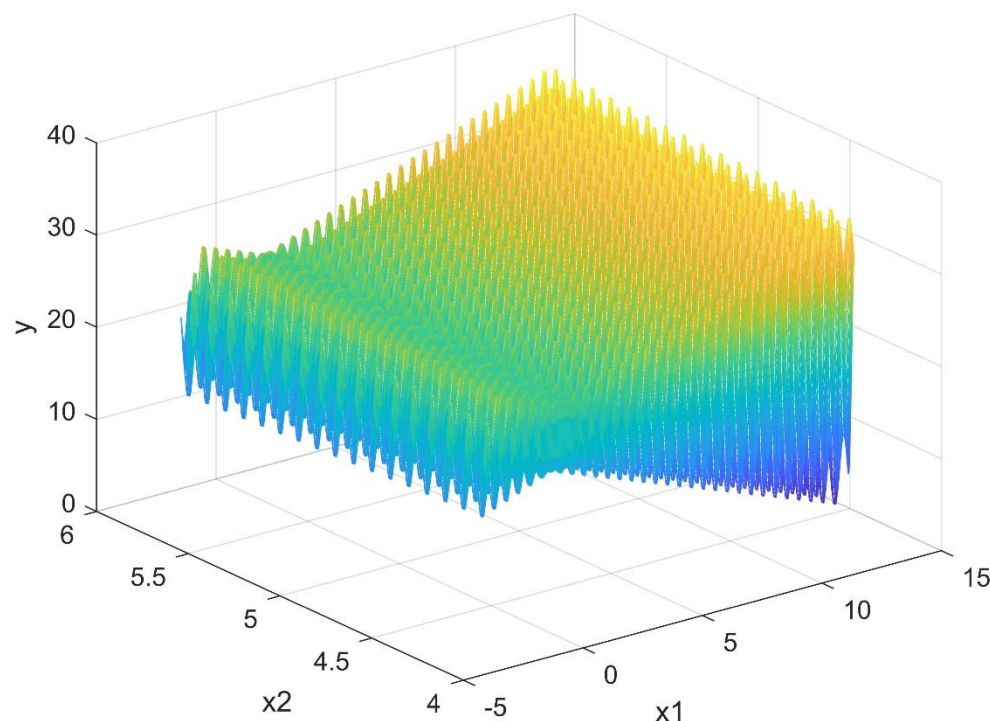
陈永刚, 牛丹梅, 范庆辉. 粒子群算法在组合优化问题上的研究与发展[J]. 电脑与电信, 2008(12):45-47.

课后作业

参考的最大值: 38.8503 提示: 自己写粒子群函数的话建议使用改进的粒子群算法, 并增加粒子个数, 多运行代码几次。

分别使用fmincon函数、你自己写的粒子群函数以及Matlab自带的粒子群函数求下面这个函数的最大值。

$$y = 21.5 + x_1 \sin(4\pi x_1) + x_2 \sin(20\pi x_2), \quad x_1 \in [-3, 12.1], \quad x_2 \in [4.1, 5.8]$$



函数图像看起来坑坑洼洼, 存在许多局部最大值

参考答案可在本节课件压缩包内找到, 尽量先独立思考后再看答案。

 数学建模学习交流