# 改进的混沌优化算法及其在函数优化中的应用

何春元,梁 伟,张建勇

(河海大学常州校区数理教学部,江苏 常州 213022)

**摘要**:针对基本混沌优化算法在求解三维以上的多维函数时不易求得全局最优解的局限性,通过引入解向量的优选,提出了一种改进的混沌优化算法,主要思路是通过多次可行解向量的混沌优选,将可行解定位到最优解的附近,再用二次载波进行搜索找出多维函数的全局最优解.仿真计算表明:该算法对三维以上函数可以显著提高搜索精度,收敛性能好,容易找到全局最优解.

**关键词**:混沌;混沌优化算法;函数优化;解向量优选

**中图分类号**:O224 **文献标识码**:A **文章编号**:1000-1980(2008)03-0423-04

混沌是一种普遍的非线性现象,其行为复杂且类似随机,但存在一定的内在规律性.混沌的发现,对科学的发展产生了深远的影响.混沌具有其独特的性质[1]:初值敏感性、遍历性、规律性,针对混沌具有遍历性这一特点,可将其作为搜索过程中避免陷入局部极小的一种优化机制,如果利用混沌变量进行优化搜索,无疑会比随机搜索更具优越性,因此,混沌已成为一种有用的优化技术,并得到广泛重视和大量研究.混沌优化分 2 个阶段进行[2]:首先,在整个空间内按混沌变量的变化规律依次考察经过的各点,接受较好点作为当前最优点;其次,一定步数后认为当前最优点已在实际最优点附近,然后以当前最优点为中心,附加一混沌小扰动,进行再搜索寻找最优点.文献[3]利用载波的方法将 Logistic 映射产生的 $n$ 维混沌变量 $t$ 引入到优化变量 $X$ 中,然后利用混沌变量进行搜索.所采用的二次载波用向量 $X^* + \alpha u$ 来改善当前的最优解 $X^*$,其中 $\alpha u$ 为一个很小的 $n$ 维混沌变量,$\alpha$ 为调节常数.这种改善对多变量的目标函数效果不明显,其原因是:一开始用多维混沌搜索后,解未必能到达最优解的附近;其次是由于 $\alpha$ 为常数,这种改善只能在次优解的单侧邻域内进行,同时在再搜索过程中无法改变扰动幅度,使得找到最优解的精度并不高.文献[4]对文献[3]的单向不足作了改进,用 $X^* + z_t(u - 0.5)$ 来改善当前的最优解 $X^*$,其中 $u$ 为混沌变量,$z_t$ 随着搜索的进行不断地变小,$z_t(u - 0.5)$ 可正可负.

通过对一些函数的测试发现,只要函数的极值点只有 1 个,或自变量的维数不大时(一般不超过 3),文献[4]提出的改进方法还是非常有效的,但当自变量较多且目标函数的极值点较多时,寻优效果比较差.本文针对文献[3-4]算法中存在的一些不足,提出改进方法,即引入解向量的优选,将解向量尽可能定位到最优解的附近,再进行二次混沌搜索,找出全局最优解.经过仿真测试表明,这种方法有明显的优越性.

## 1 改进的混沌优化算法

### 1.1 混沌优化算法

考虑连续对象的全局最小值优化问题:

$$\min f(X) \qquad X = (x_1, x_2, \cdots, x_n) \qquad (a_k \leqslant x_k \leqslant b_k, k = 1, 2, \cdots, n) \tag{1}$$

基本算法是:先确定一个初始变量 $t$($n$ 维向量),式按(2)将其对应到可行域中的 $n$ 维变量 $X$:

$$x_k = a_k + t_k(b_k - a_k) \tag{2}$$

式中 $t_k$ 为混沌变量 $t$ 的第 $k$ 个分量.用 Logistic 映射求得一个混沌变量 $t$,再按式(2)求得变量 $X$,对每一个 $X$ 求得目标函数值 $y = f(X)$,这样重复一定的次数,从中取出对应目标函数值的最小者获得问题的解,具体算

收稿日期:2007-11-26

基金项目:河海大学常州校区创新基金(CC2007-004)

作者简介:何春元(1958—),男,江苏常州人,副教授,主要从事计算数学研究.

法步骤见文献[3].

### 1.2 混沌优化算法的改进

通过对一些测试函数的计算发现,如果自变量的个数 $n(n\leqslant 2)$ 较小时,混沌优化算法效果很好,但对较大的 $n(n>3)$,精度较差.如果自变量的个数 $n$ 较大时,直接用基本混沌优化算法不能获得最优解,但经过搜索,自变量的某些分量会到达最优解附近,而自变量(解向量)不能获得最优.下面从2个方面对基本混沌优化算法进行改进:

a. 对这种混沌搜索再重复 $m$ 次(一般可取 $m=10$),利用混沌变量对初始值的敏感性,每次搜索取不同的初始值,可以得出 $m$ 组解向量.尽管每一组解不尽相同,但解中的一些分量会接近最优解.为了把这些解向量中那些接近最优解的分量找出,将这些解向量构造成一个 $m\times n$ 的矩阵 $\boldsymbol{A}$,其目的是对矩阵 $\boldsymbol{A}$ 中的每一列选择一个分量构成一个新的向量,代入目标函数中进行计算,使得目标函数值更优.这种新向量的选择,仍然可以采用混沌变量来实现.

例如:对 $n=7$,初始值取一随机数(比如 $v=0.187\,901\,148\,530\,57$),取小数点后7位数字再加1(以保证这些数字是矩阵 $\boldsymbol{A}$ 的行数),对应的选择为2,9,8,10,1,2,2,即依次取矩阵 $\boldsymbol{A}$ 中每一列的第2行,第9行……构成一个新向量为 $\boldsymbol{X}=(a_{21},a_{92},a_{83},a_{104},a_{15},a_{26},a_{27})$,记此选择为

$$\boldsymbol{X}=\mathrm{select}(v) \tag{3}$$

下一个新向量的选取采用 Logistic 映射更新:$v\leftarrow 4v(1-v)=0.610\,377\,227\,645\,86$,对应的选择为7,2,1,4,8,8,3.由于混沌迭代具有遍历性的特点,这样经过反复多次选择,将尽可能定位到最优解的附近,称此过程为解向量的优选.

b. 对当前最优解 $\boldsymbol{X}_m$ 的每一个分量 $x_k$,用 $x_k^*+z(t_k-0.5)$ 进行调整,使混沌搜索在次优解的双侧邻域内进行.这里 $x_k^*$ 为 $\boldsymbol{X}^*$ 的第 $k$ 个分量,$t_k$ 为混沌变量 $\boldsymbol{t}$ 的第 $k$ 个分量,$z$ 的初值取为

$$z=\min(x_k^*-a_k,b_k-x_k^*)\alpha \qquad (\alpha\in(0,0.5)) \tag{4}$$

同时在迭代过程中将 $z$ 的值进行更新:$z\leftarrow\lambda z$,$\lambda\in[0.9,0.999]$.最终找出全局最优解.

### 1.3 改进的混沌优化算法步骤

第1步,初始化.置计数器 $\gamma=0$,随机选取一个 $n$ 维向量 $\boldsymbol{t}=(t_1,t_2,\cdots,t_n)$ 作为混沌变量的初值,其分量在区间[0,1]内,初始目标函数值为 $y^*=f(\boldsymbol{X}^*)$,其中 $\boldsymbol{X}^*$ 为 $n$ 维向量,它的第 $k$ 个分量 $x_k^*$ 由式(2)决定.

第2步,构造矩阵 $\boldsymbol{A}$.(a)更新计数器 $r\leftarrow r+1$.如果 $r>10$,完成矩阵 $\boldsymbol{A}$ 的构造,转到第3步进行向量的优选,否则作如下混沌搜索:设置迭代次数 $N$,置搜索次数 $j=1$.(b)用 Logistic 映射产生新的混沌变量:$t_k\leftarrow 4t_k(1-t_k)$($k=1,2,\cdots,n$),按式(2)求出向量 $\boldsymbol{X}$,计算目标函数值 $y=f(\boldsymbol{X})$.(c)如果 $y<y^*$,则更新最优值:$\boldsymbol{X}^*=\boldsymbol{X}$,$y^*=y$.(d)置 $j\leftarrow j+1$,如果 $j>N$,则矩阵 $\boldsymbol{A}$ 的第 $r$ 行取为当前搜索的最优解 $\boldsymbol{X}^*$,转到(a),否则转到(b)继续进行搜索.

第3步,解向量的优选.(a)对矩阵 $\boldsymbol{A}$,设置优选次数 $G$,随机取一个[0,1]内的混沌初始值数 $u$,置计数器 $r=1$.(b)根据 $u$ 按式(3)得到对应的选择 $\boldsymbol{X}$,计算目标函数值 $y=f(\boldsymbol{X})$.(c)如果 $y<y^*$,则更新最优值:$\boldsymbol{X}^*=\boldsymbol{X}$,$y^*=y$.(d)用 Logistic 映射产生混沌变量:$u\leftarrow 4u(1-u)$,更新计数器 $r\leftarrow r+1$.如果 $r>G$,获得当前的最优解 $\boldsymbol{X}^*$.转到第4步,否则转到(a).

第4步,二次载波.(a)按式(4)给定一个初始值 $z$ 和混沌变量初值 $\boldsymbol{t}=(t_1,t_2,\cdots,t_n)$、小的正数 $\varepsilon$ 和 $\lambda$.(b)$x_k=x_k^*+z(t_k-0.5)$($x_k^*$ 为 $\boldsymbol{X}^*$ 的第 $k$ 个分量,$t_k$ 为混沌变量).(c)计算目标函数值 $y=f(\boldsymbol{X})$,如果 $y<y^*$,则更新最优值:$\boldsymbol{X}^*=\boldsymbol{X}$,$y^*=y$,同时更新 $z$ 值:$z=\lambda z$,当 $z<\varepsilon$ 时输出结果,计算结束,否则转到(a)重新进行搜索.

## 2 实例仿真测试

下面用具体实例进行仿真测试来说明改进的混沌优化算法(以下简称本文算法)在函数优化中的应用.考虑以下6个变量的函数:

$$f(\boldsymbol{X})=\varphi(x_1)+\varphi(x_2)+\cdots+\varphi(x_6) \tag{5}$$

这里 $\varphi(x_k)=\sin(20x_k)+\dfrac{1}{x_k+2}(0\leqslant x_k\leqslant 1)$,单变量函数 $y=\varphi(x)$ 的图形如图1所示.

从图 1 可知,函数 $y=\varphi(x)$在区间[0,1]内有 3 个极小值点 $x_1\approx -0.553$, $x_2\approx -0.608$, $x_3\approx -0.651$,这 3 个极小值很接近,其最优值为 $\varphi(x_3)\approx -0.651$(图 1 中虚线交叉处),因而函数 $f(X)$的最优值为 $f(x_3,x_3,\cdots,x_3)\approx -3.90$.

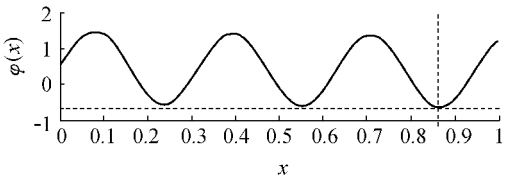


图 1　函数 $y=\varphi(x)$的图形

Fig.1　Diagram of $y=\varphi(x)$

取迭代次数 $N=50$, $n=6$,每次随机取一个初始值,按算法第 1 步和第 2 步进行搜索得到表 1 所示的 10 次结果(最后一列为对应的函数值,并按最后一列作了从小到大的排序).

从这 10 次的计算中可以发现,其解只有个别分量(由黑体表示的数据)接近最优解. 按算法第 3 步对这 10 组解实施一次解向量的优选,取 $G=5\,000$,得到一组解为

$$X^*=(0.8555,0.8767,0.8582,0.8540,0.8638,0.8764)\qquad y^*=-3.7629$$

这个解对应的就是表 1 中带有方框的数据,得到的最优值 -3.7629 与精确值 -3.90 较为接近(在没有优选时,10 次中接近的值是 -3.0696).

一般情况下,经一次解向量优选未必能得到满意的结果. 在实际计算中可以对算法第 3 步重复多次,表 2 是对表 1 中的 10 组解重新实施 10 次解向量优选得出的结果(最后一列为对应的函数值,并按最后一列作了从小到大的排序).

表 1　10 次搜索结果

Table 1　Results from 10 searches

| 序号 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $y$ |
|---|---|---|---|---|---|---|---|
| 1 | 0.5264 | 0.8776 | 0.2472 | **0.8540** | 0.5941 | 0.2374 | **-3.0696** |
| 2 | **0.8555** | 0.2561 | 0.8439 | 0.2316 | 0.8873 | 0.1947 | -3.0005 |
| 3 | 0.5746 | 0.8131 | **0.8582** | 0.8866 | 0.8491 | 0.8376 | -2.9745 |
| 4 | 0.5576 | 0.8947 | 0.9223 | 0.2325 | 0.8543 | 0.8553 | -2.9354 |
| 5 | 0.5030 | **0.8767** | 0.9176 | 0.8784 | 0.8733 | **0.8764** | -2.8160 |
| 6 | 0.2390 | 0.8540 | 0.8894 | 0.1751 | **0.8638** | 0.5324 | -2.7969 |
| 7 | 0.2139 | 0.8782 | 0.2637 | 0.8592 | 0.8050 | 0.8988 | -2.5668 |
| 8 | 0.9046 | 0.8551 | 0.5303 | 0.2650 | 0.5502 | 0.2911 | -2.5149 |
| 9 | 0.2376 | 0.8439 | 0.8475 | 0.9177 | 0.2878 | 0.5253 | -2.4026 |
| 10 | 0.8948 | 0.5216 | 0.2081 | 0.8516 | 0.5643 | 0.1744 | -2.3851 |

表 2　对表 1 实施 10 次解向量优选的结果

Table 2　Results from 10 iterations of the optimization of solution vectors in Table 1

| 序号 | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $y$ |
|---|---|---|---|---|---|---|---|
| 1 | **0.8555** | **0.8776** | **0.8582** | **0.8592** | **0.8638** | **0.8553** | **-3.8262** |
| 2 | 0.8555 | 0.8776 | 0.8582 | 0.8592 | 0.8638 | 0.8553 | -3.8262 |
| 3 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8553 | -3.7763 |
| 4 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 5 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 6 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 7 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 8 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 9 | 0.5576 | 0.8767 | 0.8582 | 0.8540 | 0.8638 | 0.8764 | -3.7629 |
| 10 | 0.5576 | 0.8551 | 0.8475 | 0.8516 | 0.8638 | 0.8553 | -3.7311 |

优选得到的最好结果为

$$X^*=(0.8555,0.8776,0.8582,0.8592,0.8638,0.8553)\qquad y^*=-3.8262$$

最后再通过算法第 4 步二次载波,取 $\alpha=0.001$, $\lambda=0.999$,混沌搜索的最后结果为

$$X^*=(0.8643,0.8645,0.8643,0.8645,0.8642,0.8642)\qquad y^*=-3.9051$$

$-3.9051$ 已达到了理论上的最优值.

为了进行比较,取函数 $f(X)=\sum_{k=1}^{n}\varphi(x_k)$, $g(X)=\sum_{k=1}^{n}\mathrm{int}(x_k)$,式中 $X=(x_1,x_2,\cdots,x_n)$ 为 $n$ 维变量, $\varphi(x_k)=\sin(20x_k)+\dfrac{1}{x_k+2}(0\leqslant x_k\leqslant 1)$, $\mathrm{int}(x_k)$ $(|x_k|\leqslant 5.12)$ 是取 $x_k$ 的整数部分,当 $n=6$ 时 $f(X)$ 就是式(5),当 $n=5$ 时 $g(X)$ 就是文献[3] 中的函数 F3. 对 $n=2,3,4,5,6$ 分别用文献[3-4] 中的算法和本文算法计算(对每个 $n$ 重复计算 100 次),结果见表 3.

表 3　不同算法求解算例的结果比较

Table 3　Comparison of the calculation results from different methods

| 算　法 | 自变量个数 $n$ | 目标函数 $g(X)$ | | 目标函数 $f(X)$ | |
|---|---|---|---|---|---|
| | | 达到最优解次数 | 平均运算时间/s | 达到最优解次数 | 平均运算时间/s |
| 文献[3]算法 | 2 | 100 | 0.0461 | 88 | 0.0852 |
| | 3 | 94 | 0.0477 | 21 | 0.0844 |
| | 4 | 33 | 0.0483 | 3 | 0.0851 |
| | 5 | 3 | 0.0490 | 0 | 0.0869 |
| | 6 | 1 | 0.0495 | 0 | 0.0897 |
| 文献[4]算法 | 2 | 100 | 0.0450 | 97 | 0.0811 |
| | 3 | 84 | 0.0469 | 35 | 0.0831 |
| | 4 | 23 | 0.0462 | 14 | 0.0839 |
| | 5 | 9 | 0.0472 | 2 | 0.0861 |
| | 6 | 5 | 0.0480 | 1 | 0.0886 |
| 本文算法 | 2 | 100 | 0.4644 | 100 | 0.4816 |
| | 3 | 100 | 0.4781 | 100 | 0.4986 |
| | 4 | 99 | 0.4795 | 99 | 0.5009 |
| | 5 | 99 | 0.4934 | 99 | 0.5139 |
| | 6 | 97 | 0.5083 | 98 | 0.5381 |

从表3比较可知,当自变量个数较少时,3种算法都能搜索到最优解.当自变量个数较多时,文献[3-4]算法难以求得目标函数的最优解,其原因是当自变量个数较多时,$g(X)$和$f(X)$的局部最优解很多,这2种混沌优化算法在搜索过程中容易陷入局部最优解.

## 3 结 语

本文提出了用于求解多维函数优化问题的改进混沌优化算法,针对文献[3-4]中的混沌优化方法对三维以上函数优化所存在的局限性,通过引入解向量的优选进行了改进,改进后的算法能进一步改善算法的效能,提高了求得全局最优解的计算精度,容易找出全局最优解.该算法为求解多维函数优化问题提供了一种有效方法.

参考文献:

[1] 张化光,王智良,黄伟.混沌系统的控制理论[M].沈阳:东北大学出版社,2003.
[2] 王凌,郑大钟,李清生.混沌优化方法的研究进展[J].计算技术与自动化,2001,20(1):1-5.
[3] 李兵,蒋慰孙.混沌优化方法及其应用[J].控制理论与应用,1997,14(4):613-615.
[4] 赵强.改进的混沌优化方法及其应用[J].自动化与仪器仪表,2006,125(3):90-92.
[5] 崔畅,赵强.改进的混沌优化算法研究[J].科学技术与工程,2007,7(3):307-310.
[6] 郑肇葆.混沌映射在优化计算中的应用[J].武汉大学学报:信息科学版,2007,32(11):998-1000.
[7] 廖艳芬,马晓茜.改进的混沌优化方法在电站机组负荷分配中的应用[J].动力工程,2006,26(1):93-96.
[8] 尤勇,王孙安,盛万兴.新型混沌优化方法的研究及应用[J].西安交通大学学报,2003,37(1):69-72.
[9] 杨迪雄,李刚,程耿东.非线性函数的混沌优化方法比较研究[J].计算力学学报,2004,21(3):257-262.
[10] 李金屏,韩延彬,孙志胜.混沌优化算法的性能分析[J].小型微型计算机系统,2005,26(8):1340-1344.

# Modified chaos optimization algorithm and its application in function optimization

**HE Chun-yuan, LIANG Wei, ZHANG Jian-yong**

(*Department of Mathematics and Physics, Hohai University, Changzhou* 213022, *China*)

**Abstract**: Due to the difficulties of finding global optimization solutions to multi-dimensional functions using the chaos optimization algorithm, a modified chaos optimization algorithm is developed based on the optimization of solution vectors. With several iterations of optimization, the solution vectors approach the optimal solution, and the global optimal solutions to the multi-dimensional functions can be found by the carrier wave two times. Simulation results show that this method significantly improves the search accuracy and convergent behavior and is effective in finding global optimal solutions to multi-dimensional functions.

**Key words**: chaos; the chaos optimization algorithm; function optimization; solution vector optimization