Computer Science and Application 计算机科学与应用, 2012, 2, 221-226
http://dx.doi.org/10.12677/csa.2012.24039 Published Online October 2012 (http://www.hanspub.org/journal/csa.html)

Gene Expression Programming with Error Remainder*

Jiali He[1], Shibin Xuan[2]

[1]School of Mathematics and Information Science, Yulin Normal University, Yulin
[2]College of Mathematics and Computer Science, Guangxi University for Nationalities, Nanning
Email: hjljuly@163.com

Received: Aug. 14th, 2012; revised: Aug. 29th, 2012; accepted: Sep. 7th, 2012

Abstract: The traditional GEP is superior over other algorithms for function mining, but it gets easily struck at local optimal solution and jump difficultly out of local optimal solution, and result in greater absolute error. A new GEP with remainder is proposed. When the individuality fall in local peak, the error between function value and target value is regarded as remainder term in proposed method. The remainder is considered as a new target value and the algorithm is keeping on working to search the new function. This process is repeated until the error reaches desired target. The absolute value of remainder which means function absolute error has been proved to converge to 0 in probability. The simulation results show that the algorithm can reduce absolute error efficiently.

Keywords: GEP; Absolute Error; Remainder Term; Approximation; Accuracy

带误差余项的基因表达式编程*

何家莉[1]，宣士斌[2]

[1]玉林师范学院数学与信息科学学院，玉林
[2]广西民族大学数学与计算机科学学院，南宁
Email: hjljuly@163.com

收稿日期：2012 年 8 月 14 日；修回日期：2012 年 8 月 29 日；录用日期：2012 年 9 月 7 日

摘　要：基因表达式算法(Gene Expression Programming, GEP)在函数挖掘方面有较大的优越性，但容易过早收敛，且很难跳出局部最优解，导致最后的绝对误差较大，为此提出了一种带有余项误差的方法。该方法在当个体陷入局部最优时，将函数值与目标值的误差作为余项，进一步把该余项数据再进行函数挖掘，由此找到的新函数更接近目标值。反复多次用 GEP 寻找跟余项数据逼近的函数，再计算新的余项，可让余项的绝对值也就是函数的绝对误差依概率收敛于 0。并从理论上证明了这种算法的可行性。通过仿真例，结果表明该方法在降低绝对误差上起到良好的作用。

关键词：基因表达式编程；绝对误差；余项；逼近；精度

1. 引言

2000 年 Ferreira 提出了一种新的遗传算法——基因表达式编程[1-3]。它作为一种基于基因型组和表现型组的新型进化算法，它结合了遗传算法(GA)和遗传规划(GP)的优点。GEP 的个体是由多个长度固定的基因组成的线性字符串，这些个体在进行计算时被解码成表达式树(Expression Trees, ET)。其搜索空间为线性串，其解空间为表达式树，这种搜索空间和解空间分离的特点保证了 GEP 的性能优于基本的 GA。从功能上，GEP 和 GP 类似，能发现和揭示复杂问题内在的规则。在解决复杂问题时，比传统的 GP 效率高 2~4 个数量级。

我们把 GEP 应用到函数挖掘、符号回归、参数优

*基金项目：玉林师范学院青年项目(2009YJQN100)。

化、时间序列预测等方面，可以看出虽然基本的 GEP 可以得到较好的结果，但是仍然有着明显早熟(过早收敛)的缺点。在函数挖掘上对那些数据点少，且数值小，目标函数又是显函数的时候 GEP 还能发挥其优势。一旦数据点过多，数值较大，目标函数是隐函数的时候，基本的 GEP 就无能为力了。用 GEP 寻找出的函数其绝对误差非常大，这时的 GEP 基本上是不收敛的。对这类数据论文[4,5]将 GEP 和克隆选择算法相结合，论文[6]添加模拟退火机制其效果都是增加种群在进化时选择的多样性，但没有增加算法的导向性，使得算法运行时间大大增加且陷入局部最优后很难跳出。论文[7]研究的小生境 GEP 算法可以使种群暂时跳出局部最优，但是又会立刻陷入另一个局部最优，最后陷入在一个个局部中，难以找到全局最优。论文[8]以牺牲时间为代价增大搜索空间在函数优化上还适用，但在函数挖掘上仍没有解决如何跳出局部最优，极容易陷入死循环，且可以通过在进化中增加新种群或者增加基因个数能达到同样效果。论文[9]解决了如何使得初始种群多样化且个体优秀的问题，但在寻找函数时没有导向性，初始阶段很难判断个体谁优谁劣。随着迭代的深入，多样化的种群慢慢会变得类似。论文[10]增加了一个算子，对早熟问题没有起到多大帮助。总结以上的论文我们归结为一点，算法都缺乏导向性。因此以上论文对 GEP 无论怎么改进都还是极容易陷入局部最优，且很难跳出。迭代非常多次后得出的函数与目标函数的绝对误差值仍然很大。许多改进的 GEP 算法也只能把这个绝对误差在量上减小，无法在质上取得飞跃。因此，本文针对这类问题提出了带有余项的 GEP 改进算法，给算法带来了一些导向性，并对其做了理论分析和数值试验，在同样的运算时间内检验效果其绝对误差大大减小，其结果远优于其它算法。

2. 基因表达式编程中的余项

在多次运行 GEP 算法时会发现，对于某些数据逼近到一定程度时误差就很难缩小，即使改进算法也只能略微缩小一点，无法有实质性突破。我们用 GEP 算法找出的函数 $f_0(X)$ 与目标值 Y_0 仍存在有较大差距，也就是绝对误差较大，但是种群已经很难进化了。这时我们用 $f_0(X)-Y_0=Y_1$ 得到一个余项[11,12]。再把余项 Y_1 当成是一个目标数据，重新用 GEP 继续逼近与余项目标值配对的函数 $f_1(X)$。这样寻找到的函数列与目标函数值之差的余项会越来越小。当余项接近 0 时，此算法就找到最终的结果。

设 Y_0 是第一次给出目标函数中实际测量的数据，$Y_i(i=1,2\cdots n)$ 是后面的余项。我们有如下的函数 $f_0(X)$、$f_1(X)\cdots f_n(X)$，$X=(x_1,x_2\cdots x_s)$。其中 $f_i(X)(i=1,2\cdots n)$，是对余项 $Y_i=\left(y_1^i,y_2^i\cdots y_m^i\right)^T, i=1,2\cdots n$ 做的逼近函数。对于第一次给出的实际数据 Y_0，我们用 GEP 寻找相应的函数为 $f_0(X)$，余项为 $f_0(X)-Y_0=Y_1$。接着再用 $f_1(X)$ 逼近目标数据 Y_1，得到的余项为 $f_1(X)-Y_1=Y_2$，后面的函数余项分别为 $f_2(X)-Y_2=Y_3,\cdots f_n(X)-Y_n=Y_{n+1}$。即 $f_0(X)-Y_1=Y_0$，$Y_1=f_1(X)-Y_2,\cdots Y_{n-1}=f_{n-1}(X)-Y_n$，把这些等式联立起来：

$$f_0(X)-f_1(X)-f_2(X)-\left[f_3(X)\cdots\left[f_n(X)-Y_{n+1}\right]\right]=Y_0$$

去括号，

$$f_0(X)-f_1(X)+f_2(X)-f_3(X)+\cdots+(-1)^{n-1}f_{n-1}(X)+(-1)^n\left[f_n(X)-Y_{n+1}\right]=Y_0$$

令

$$g_n(X)=f_0(X)-f_1(X)+f_2(X)-f_3(X)\cdots(-1)^n f_n(X),\ n=0,1\cdots\infty \quad \text{①式}$$

经过这样辗转相减及合并，最后由 GEP 算法找出的 $g_n(X)$ 与目标数据的绝对误差趋向于很小。在本文中用余项的绝对误差之和判断 GEP 逼近的效果，把 Y_{i+1} 的 m 个数据点中每一项误差的绝对值求和，即

$$\Delta=\sum_{j=1}^{m}e_j=\sum_{j=1}^{m}\left|f_i\left(X_j\right)-y_j^i\right| \quad \text{②式}$$

显然误差 Δ 越小函数逼近得越好，那么 Δ 是否趋向于 0 呢？有下面的定理给予了解释。

3. 算法收敛定理

定理：当 n 趋于无穷大时，$g_n(X)$ 依概率收敛于目标函数。

证明：命题是要证明 $\lim_{n\to\infty}P\left(\left|g_n(X)-Y_0\right|>\varepsilon\right)=0$，

只要证明 $\lim_{n\to\infty}P\left(\left|Y_{n+1}\right|>\varepsilon\right)=0$ 即可。因为 $f_i(X)-Y_i=Y_{i+1}$，有 $\left|f_i(X)-Y_i\right|=\left|Y_i-f_i(X)\right|=\left|Y_{i+1}\right|$。由论文[2]定理 4.4 知 GEP 依概率收敛于全局最优，但不是强收敛到全局最优，有可能陷入局部最优。因此，每次迭代产生的最优函数 $f_i(X)$ 依概率收敛于 Y_i。对于逼近到 Y_i 的函数 $f_i(X)$，即可取到 $\left|Y_{i+1}\right|=\left|Y_i-f_i(X)\right|<\left|Y_i\right|$。而对无法逼近 Y_i 的函数 $f_i(X)$，即陷入局部最优难以进化时，我们仍可以取 $f_i(X)=0$，得 $\left|Y_i\right|=\left|Y_{i+1}\right|$。所以无论 $f_i(X)$ 是否陷入局部最优，我们都有 $\left|Y_{i+1}\right|\le\left|Y_i\right|$。又因为 GEP 始终依概率收敛于全局最优，所以当序列无限时，满足 $\left|Y_{i+1}\right|<\left|Y_i\right|$ 的 i 必定有无限个。因为余项 $\left|Y_i\right|$ 有下确界 0，且数列 $\left|Y_i\right|$ 总体单调递减，所以 $\lim_{n\to\infty}P\left(\left|Y_n\right|>\varepsilon\right)=0$。

当 Y_n 依概率收敛到 0 时有，对于 $\forall\varepsilon>0$，$\exists N\in N_+$，能找到一个 N，使得 $\forall n>N$，有 $\left|Y_{n+1}\right|<\varepsilon$，即

$$\left|\begin{matrix}f_0(X)-f_1(X)+f_2(X)-f_3(X)\cdots\\+(-1)^n\left[f_n(X)-Y_0\right]\end{matrix}\right|<\varepsilon$$

也就是

$$\begin{aligned}g_n(X)=&\,f_0(X)-f_1(X)+f_2(X)\\&-f_3(X)\cdots(-1)^n\,f_n(X)\end{aligned}$$

为要找的逼近 Y_0 的函数。

4. 带余项的基因表达式编程算法

本文把余项分析及一些论文改进的 GEP 算法有效的结合起来，提出了带余项的基因表达式编程算法。算法的具体步骤如下：

1) 确定 GEP 算法的各个参数，其中包括变异概率，变叉概率，倒置概率，重组概率等。并设定停机条件，可以设相对误差或者绝对误差下限。本文采用余项的绝对误差和也就是用②式判定收敛与否。然后转向 2)

2) 随机生成N组种群，并确定目标数据值 Y_i。然后转向 3)。

3) 计算种群的适应度，采用绝对误差的适应度函数 $\text{fitness}=\sum_{j=1}^{m}\left(M-\left|f_i\left(X_j\right)-y_j^i\right|\right)$，其中 M 表示选择的范围，视不同问题而取值。再根据适应度进行最优选择，即保留最优种群，其它种群用转轮盘选择。然后转向 4)。

4) 进行变异，插串，倒置，重组等算子操作。当种群迭代到设定的次数就进行一次全局变异，设定变异率为 100%同时加入迁徙算子[6]或者只保留最优个体，删除其余个体，并用随机生成的种群取代。再转向 5)。

5) 判断种群是否达到设定精度，如果结果达到设定精度则退出循环并输出结果。如果种群经过一定次数的迭代后难以继续进化(可以设定达到一定的迭代次数后最优种群仍没有变化)且同时又没达到设定精度时，退出本阶段迭代，记录这次的最优种群，并解码成函数 $f_i(X)$。用 $f_i(X)$ 减去目标值 Y_i 得到误差余项 Y_{i+1}。再根据余项 Y_{i+1} 作为新的目标数据值更新 Y_i，然后转向 2)又开始执行整个 GEP 算法接着寻找 $f_{i+1}(X)$。如此循环直到找到的某个函数 $f_n(X)$ 与目标值的余项绝对误差达到设定的精度后停止。最后找到的一组函数 $f_i(X)\ (i=0,1,\cdots n)$，由①式得要求出的函数 $g_n(X)$。

通过不断的循环迭代，开始由一个较大的余项误差慢慢的迭代到一个较小的余项误差，这样逐渐把余项的误差趋向于 0，就保证了即使种群过早收敛也能找到满意的解。详细的运算过程见流程图 1。

5. 数值试验

例 1：有如下数据，见表 1。根据这些数据寻找最接近的函数，本文例子均在 matlab 中完成。数据的精确解是：$xy+2^x-2^y=0, y\in[-300,0]$，其中 x 是自变量，这是一个隐函数。

该数值试验用本文算法总共循环运行七次 GEP 算法，每执行一次 GEP 算法直到种群难以进化后才停止并转到下一次 GEP，其中这七次运行迭代为 3000~4000 次之间。最后第七次得到的余项误差为 0.4164。GEP 设定的具体参数见表 2，运行算法次数与余项误差之间的关系见表 3。其它文献中的算法经过数次试验取最好绝对误差效果见表 4，并且迭代次数都是十万次，时间比本文算法的长 8 倍左右。

从表 3 看出，本文算法每次寻找的余项的绝对误差和都是在逐渐减小，当算法运行到第 7 次时，绝对误差为 0.4164 时达到满意解。而其他算法在迭代到很大次数时仍然未有收敛的迹象，说明本文算法对 GEP

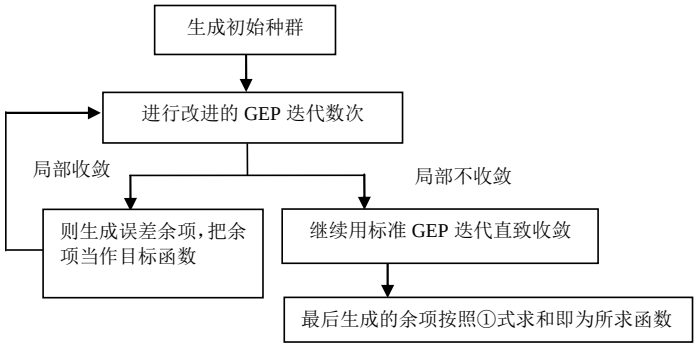

Figure 1. Algorithm flow
图 1. 算法流程

Table 1. Known data and value of function
表 1. 已知数据点对应的函数值

X	0	1	2	3	4	5	6	7	8	9	10
Y	0	–1.6902	–1.8624	–2.6123	–3.9841	–6.3977	–10.6666	–18.2858	–31.9999	–56.8888	–102.4000

Table 2. Parameter of GEP in example 1
表 2. 例 1 中 GEP 算法参数

种群大小	函数符集	终结符集	头部长度	基因个数	连接符	交叉概率	变异概率
40	{+，–，*，/,Pf,sqrt ,exp,pow2}	{x}	6	5	+	0.3	0.05

Table 3. Absolute error of remainder term and times in this paper's GEP
表 3. 本文算法运行的次数与余项的绝对误差

运行序数	1st	2nd	3rd	4th	5th	6th	7th
余项绝对误差	89.4288	33.7167	11.6531	4.9738	2.2261	0.9362	0.4164

Table 4. Absolute error of remainder term in other algorithms
表 4. 其它算法余项的绝对误差

算法	传统 GEP	小生境 GEP[7]	文献[8]GEP	文献[9]GEP
余项绝对误差	84.2451	70.2453	75.0148	65.1785

有着明显的改进。这 7 个函数在 matlab 中分别显示是：(其中 Pf 表示平方，sqrt 表示开方，exp 表示 e 的几次方，pow2 表示 2 的几次方)

$f_0(x)$ = (((((cos((Pf(x)) – 134.5642)) – (log(exp(x))))
+ ((cos(x)).∗((131.7114./58.0034).∗x)))
+ ((exp(103.8348./103.8348)) – (log(exp(x)))))
+ ((cos((Pf(x)) – 103.3699)) – (log(exp(x)))))
+ ((cos((Pf(x)) – 103.3699)) – (log(exp(x))))

$f_1(x)$ = ((((log((sqrt(146.7804)) – (sqrt(145.4349))))
+ (Pf((Pf(cos(127.7232))./((cos(120.393)).x))))
+ (x.∗(cos(75.9632.∗(x – 123.7387)))))
+ (x.∗(cos(pow2(x))))) + (x.∗(cos((x.∗82.5155).∗69.6198)))

$f_2(x)$ = (((((sin((58.2934 + 148.5415) – x))
– (cos(x – 120.0442))) + (sqrt(Pf(Pf(log(log(log(x)))))))
+ (sqrt(Pf(log(log(log(x)))))) + ((sin(cos(54.0402)))
– (cos(Pf(x))))) + (sqrt(log(x)))

$f_3(x)$ = ((((sin(log((pow2(x)) + 98.3977))) + (sin(79.48)))
+ (Pf(cos(111.6784 + ((70.256 – 140.9393) – x)))))
+ (cos(Pf(Pf((Pf(x)) + (sin(111.4544)))))))
+ (sin((sin(sin(69.2887))) + x))

$f_4(x)$ = ((((sin(141.7667 – ((cos(101.7531))./(exp(x)))))
+ (sin(141.7667 – (x./(125.4522 – x)))))
+ (cos(cos((x./72.2168) – x))))
+ (((cos(141.7667))./65.9912)./(cos(x – 143.5077))))
+ (((cos(116.245))./50.2904)./(cos(x – 75.7064)))

$f_5(x)$ = ((((cos(21.5928)) + (x./(log((18.0318.∗–12.9895).
∗(tan(8.133)))))) + (sqrt(x + (sin(exp(x))))))
+ ((tan((–42.0622 + 27.6526).∗x)).
∗(exp(tan(–13.856))))) + (x./–2.7459)

$f_6(x)$ = ((((tan(12.1602)) + (sqrt(exp(–12.3872 – (tan(x
+ 47.0742)))))) + (sin(((sin(16.3804))

– (cos(*x*))./(23.9285 – 35.2539))))
+ (log(log(log(7.2971 + 26.8556)))))
+ (log(log(log(*x* + 21.0724))))

最后求出的函数是

$$g_n(x) = f_0(x) - f_1(x) + f_2(x) - f_3(x) + f_4(x) - f_5(x) + f_6(x)$$

从表 4 可以看出，其它算法耗时比本文算法长，而绝对误差仍大许多。是因为长时间陷入到局部最优无法跳出找到全局最优，其原因已经在引言讲明，主要是由于算法本身的缺陷造成的。

例 2：已知函数 $xy + \sin z + y - 2z = 0$ 的部分数据点，其中 *x*、*y* 是自变量，这也是一个隐函数。

取 *x* 从 1，步长为 1 直到 15；取 *y* 的值分别为：1、4、6、8、10、11、12、13、14、15、14、13、12、11、10；则 *z* 的近似值分别为：1.4987、5.7428、11.5841、20.4985、29.5254、38.9790、47.7195、58.8669、70.4904、82.9819、84.2648、84.6070、84.2648、82.9819、79.5710。

用本文算法总共循环运行九次 GEP，每执行一次 GEP 算法直到种群难以进化后才停止并转到下一次 GEP。其中这九次迭代都为 3000~4000 次之间。最后得到的绝对误差为 0.4946。GEP 的种群大小为 50，终结符集为 $\{x, y\}$，其余设定的参数与例 1 一致，运行的结果与数值见表 5。表 6 是其它文献算法经过数次试验取最好的误差结果，迭代次数都是十万次，时间比本文算法的长 7 倍左右。

从表 5 可以看出余项也是在逐渐减小的，当算法运行到第九次时，绝对误差为 0.4946 达到满意解，这 9 个函数在 matlab 分别显示为：

*f*0(*x*,*y*) = ((((–25.4089.∗(sin((sin(*x*)).∗(exp(*y*)))))
+ (29.8251./(*x*./(cos(*x*))))) + ((tan(tan(–42.5343))).∗((4.239
+ 5.0885) – *x*))) + ((log((37.2277 + 48.553) + (*x*
+ 27.9716))).∗*y*)) + ((tan(*x*)).∗(*y* – (49.8421 + –25.0548)))

*f*1(*x*,*y*) = ((((sin(–42.0939.∗(*x*.∗*x*))) + ((sqrt(*y*))
– (sqrt(*x*)))) + (tan(((exp(*x*))./–24.4613) + 41.4499)))
+ (tan(((exp(*y*))./ –24.4613) + 41.4499)))
+ (sin(exp(6.357.∗*x*)))

*f*2(*x*,*y*) = (((((cos(tan(*x*))) – (cos(43.239 – *x*)))
+ (cos((tan(*x* + –45.234)) – ((*y* + –0.32417) + 36.6797))))
+ ((cos(cos(*x*.∗–21.3063))) – (cos(*y* – *x*1))))
+ ((cos(25.8049.∗*y*)) – (cos(*x* – 30.0173)))) + (tan(14.7871))

*f*3(*x*,*y*) = ((((cos((log(1.8924.∗*x*)).∗((sqrt(*x*)) + –2.7489)))
+ (((exp(log(7.373))).∗(*x*./ –31.2222))./*y*))
+ (sin(((*y*./–12.7764) – 21.1144) + (tan(*y*)))))
+ (tan(exp(((21.1673./4.3211) – *x*)
– (3.0926./–37.2416)))))
+ ((sin(*x*))./((16.256./*x*)./(sin(–11.6814))))

*f*4(*x*,*y*) = ((((((cos(*y*))./(6.7902./*y*))./*x*)
+ (((cos(*x*./*y*))./(tan(*y*)))./–11.0982)) + (((cos(*y*1
+ –8.6014)).∗(6.8313./–7.8236))./*x*))
+ (7.4172./((16.474.∗*x*).∗*x*))) + (((cos(*y*
+ 41.6725))./(6.8313 + –7.8236))./*x*)

*f*5(*x*,*y*) = (((((sqrt(sqrt(47.9053)))./((tan(*y*))
+ (*x*.∗16.5046))) + (tan(20.9977)))
+ ((sqrt(sqrt(22.295)))./((tan(*y*)) + (*x*.∗16.5046))))
+ (sin(sqrt(sqrt(log((log(4.0665)).∗*x*))))))
+ (cos(tan(cos(tan(–30.8088)))))

*f*6(*x*,*y*) = (((((cos((tan(*x*))./(cos(–26.2507))))./–7.5107)
+ ((cos((*x* + 29.1853)./(cos(–47.9561))))./–5.3512))
+ (cos(–22.0108))) + ((14.036.∗(exp(*x*))).∗(exp(–33.0892
+ *x*)))) + (cos(*x*./((*x*/–6.1815) + (18.3516 + *y*))))

*f*7(*x*,*y*) = (((((tan((sin(–34.8766))./–19.3672))./(tan(*x*
+ 18.1566))) + (sin(43.1931))) + (tan(*y*)))
+ (((sin(39.3572)).∗(cos(–13.079))).∗(cos(cos(7.2514)))))
+ ((sqrt(exp(–36.5906 – *x*))) –(tan(*y*)))

*f*8(*x*,*y*) = (((((sin(47.1824))./((cos(27.4635)) – (*y* – *x*)))
+ ((sin(47.1824))./((41.1919 – 34.4963) – *y*)))
+ ((tan((sin(–41.2997))./*x*))./(exp(*x*))))
+ (sqrt(sqrt(((exp(*y*)) – *y*)./(exp(28.1329))))))
+ ((sin(cos(exp(*y*))))./(exp(sqrt(*x*))))

Table 5. Absolute error of remainder term and times in this paper's GEP
表 5. 算法运行后的次数与余项的绝度误差

运行序数	1st	2nd	3rd	4th	5th	6th	7th	8th	9th
余项绝对误差	73.9430	22.9959	9.9490	6.8030	4.2427	3.2437	1.5637	1.0807	0.4946

Table 6. Absolute error of remainder term in other algorithms
表 6. 其它算法余项的绝对误差

算法	传统 GEP	小生境 GEP[7]	文献[9]GEP	文献[10]GEP
余项绝对误差	70.2451	45.0483	54.1589	55.9452

最后求出的函数是

$$g_n(x,y) = f_0(x,y) - f_1(x,y) + f_2(x,y) - f_3(x,y) + f_4(x,y) - f_5(x,y) + f_6(x,y) - f_7(x,y) + f_8(x,y)$$

从表 6 可以看出，其它算法耗时比本文长许多，但找到的函数的绝对误差仍然很大。

在这两个例子中，其它算法消耗的时间是本文算法的 7~8 倍，而绝度误差却是本文的 100 倍左右。经单独对小生境[7]算法用例 2 测试，使时间长达 20 倍时，其绝对误差为 40.8472 仍比本文的大 80 倍。由此说明本文算法不仅快且精度高，效果明显。

6. 结束语

对于某些数据点多，数据数值大，且数据呈现的是隐函数的情况，一般改进的 GEP 算法难以找到绝对误差很小的满意解，即使改进算法使得种群跳出局部最优，但又会立刻进入另一个局部最优。本文针对 GEP 算法在寻找复杂函数时极容易早熟，且陷入早熟后不易跳出的特点，提出了带有余项的 GEP 算法。采用文本算法，不管首次运算的误差结果有多大，都可以通过带余项的方式把每次求得的余项辗转相减，使得较大误差的余项逐渐缩小，从而找到一个较精确的解。本文从理论上分析了这种方法的可能，并证明了此方法依概率收敛，最后用数值试验验证了该方法的有效性。相比其它改进的 GEP 算法而言，大大提升了精度。

参考文献 (References)

[1] C. Ferreira. Gene expression programming in problem solving. Soft Computing and Industry Recent Applications, Springer-Verlag, 2002: 635-654.

[2] 左劼. 基因表达式编程核心技术研究[D]. 成都: 四川大学, 2004.

[3] C.-A. Yuan, C.-J. Tang, J. Zuo, F. J. Xie, A. L. Chen and J.-J. Hu. Function mining based on gene expression programming—Convergency analysis and remant guided evolution algorithm. Journal of Sichuan University (Engineering Science Edition), 2004, 36(6): 100-105.

[4] V. I. Litvinenko, P. I. Bidyuk, J. N. Bardachov, et al. Combinating clonal selection algorithm and gene express ion programming for time series prediction. Sofia: IEEE Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Application, 2005: 133-138.

[5] V. Karakasis, A. Stafylopatis. Data mining based on gene expression programming and clonal selection. Vancouver: IEEE Congress on Evolutionary Computation, 2006: 514-521.

[6] S. W. Jiang, Z. H. Cai, D. Zeng, et al. Gene expression programming based on simulated annealing. IEEE, 2005, 2: 1264-1267.

[7] 陈云亮, 李欣等. 用于关联规则挖掘的一种基于小生境技术的 GEP 算法[J]. 计算机科学, 2009, 36(11): 224-227.

[8] 向勇, 唐常杰等. 基于内嵌基因表达式编程的函数优化[J]. 四川大学学报(工程科学版), 2010, 42(4): 91-96.

[9] 胡建军, 彭宏. 基因表达式编程中的精英个体产生策略[J]. 华南理工大学学报(自然科学版), 2009, 39(1): 102-105.

[10] 莫海芳, 李康顺. 基因表达式编程的一种新遗传算子[J]. 计算机工程与应用, 2011, 47(7): 23-29.

[11] 李庆扬, 王能超. 数值分析[M]. 北京: 清华大学出版社, 2008.

[12] 刘玉琏, 傅沛仁. 数学分析讲义下册[M]. 北京: 高等教育出版社, 2001.