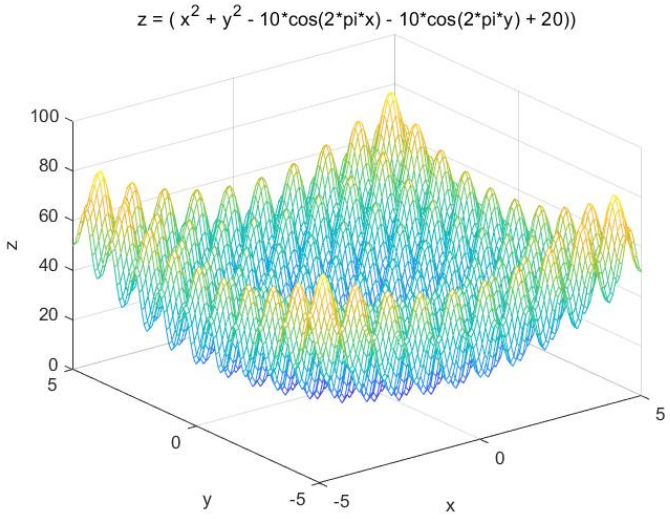
人工智能实验报告01使用遗传算法解方程

116132019089 沈仕越

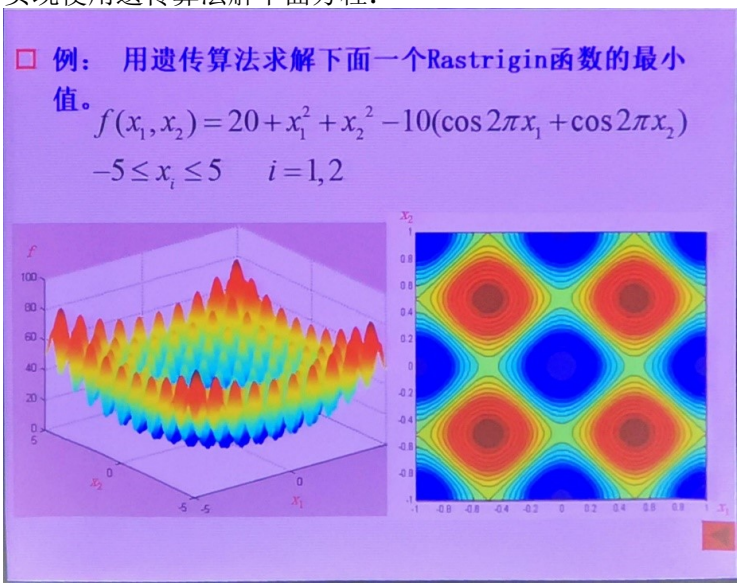
2021 年 4 月 11 日



目录

1 实验目的

实现使用遗传算法解下面方程：



2 实验平台

1. 操作系统：Microsoft Windows 10
2. 编辑器：Microsoft vscode
3. 编程语言：C++
4. 编译器：g++
5. 编译标准：C++11

3 实验过程

遗传算法是一种搜索算法，基本思想是使用计算机模拟生物进化的过程，并通过这种模拟方式在搜索域中寻找出最优解。

遗传算法将问题的解抽象成为基因（或称染色体），并让这些基因按照一定规则进行交叉、突变、自然选择等，使种群向更优解进化。

进化的过程即是迭代的过程，每一次迭代，算法都会根据某个**适应性函数**计算出每个个体的适应度，并根据适应度选择一部分个体繁衍后代，通过自然选择和突变产生新的种群，不断迭代，不断进化。

要实现遗传算法，就应该从以下一些方面考虑实现的策略，并依此编写代码。

编码的方式 是基本问题，编码的方式不同，将把问题的解映射到不同的解空间内。

传统的编码方式以二进制编码为主，但也有实数编码的方式。而本函数的解的形式是一个坐标，取值在区间$[-5,5]$上。因此可以考虑直接使用实数编码，设计基因类Chromosome包含x1，x2两个成员，取值在-5到5之间。

```
1  class Chromosome {
2  private :
3      double _x1, _x2;
4
5      // This variable only meaningful when there are multiple chromosomes
6      // needed to produce next generation.
7      // Set to 0 by default.
8      double _fit;
9
10     static std::default_random_engine _engine;
11
12 public :
13     Chromosome();
14     Chromosome(double x1, double x2);
15     Chromosome(const Chromosome & obj);
16     ~Chromosome();
17
18     // Two in, two out.
19     static std::vector<Chromosome>
20         crossover(Chromosome & c1, Chromosome & c2);
21
```

```
22      void random();
23
24      // Use random as mutation.
25      void mutate();
26
27      double getX1() const;
28      double getX2() const;
29      double getFit() const;
30
31      void setFit(double rate);
32
33      void print();
34
35      void writeX1(std::ofstream & output_file);
36      void writeX2(std::ofstream & output_file);
37  };
```

由于使用了实数编码，后续的交叉和变异的方法也应当做适当的调整，使得交叉和变异的过程科学合理。

群体设定 主要包括了初始种群的产生和种群规模的确定两个方面。这一次的实验考虑随机生成20个个体，并将种群规模控制在100个个体以内。

适应性函数 将计算出个体的适应度，并以此作为遗传（交叉）操作的依据。适应度是评价个体的标准，个体适应度越高，被选择的概率越高，被淘汰的概率越低，反之，则被选择的概率越低，被淘汰的概率越高。

本问题欲求解函数的最小值，是一个最小化问题，不妨将待求解的函数的倒数作为适应度函数：

$$\begin{aligned} Fit(f(x)) &= \frac{1}{f(x)} \\ &= \frac{1}{20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2)} \end{aligned}$$

选择的策略 决定了哪一些个体会获得繁殖的机会，需要注意，不能总挑选最好的个体，否则遗传算法将成为确定性优化方法，使种群过快收敛到

局部最优解；也不能随机选择，否则将需要长时间收敛甚至不能收敛。

个体选择概率分配方法 即根据适应度决定个体被选择的概率，这里不妨选用基础的适应度比例方法，即个体i被选择的概率为：

$$P_{si} = \frac{f_i}{\sum_{i=1}^{M} f_i}$$

选择的过程在迭代的过程中进行，代码将构造一个基因集ChromosomeSet类，它将具有一个函数，函数内实现了基因迭代的算法。具体代码较长，见附录。

选择个体的方法 即根据概率，选择进行交叉（繁殖）的个体的方法。此处选择**轮盘赌选择方法**，即按照个体顺序求出每个个体的累计概率，产生一个随机数，根据随机数落入的区域选择相应的个体。

交叉的策略 将决定两个进行交叉的个体以何种方式产生子代。两个基因交叉时，他们将相互混合产生一对新的基因。

由于先前采用了实数编码，因此二进制编码使用的**一点交叉**和 **二点交叉**都无法直接使用。

此处采用的交叉方法是，随机产生一个权重p，亲代基因分别乘上p和1-p，再将结果相加，作为子代的基因。面通过数学证明说明这种交叉方法是用何种方式让子代获得亲代的基因特征的。

设p为随机变量且$p \sim U[0,1]$

$$\text{即} f_p(x) = \begin{cases} 1, 0 \le x \le 1 \\ 0, \text{其他} \end{cases}$$

$$F_p(x) = \begin{cases} x, 0 \le x \le 1 \\ 0, x < 0 \\ 1, x > 1 \end{cases}$$

$$
\begin{aligned}
&\forall a > b, a, b \in [-\infty, +\infty] \\
&\quad 令 Y = ap + b(1-p) \\
&\qquad 则 F_Y(y) \triangleq PY \le y \\
&\qquad\qquad = P\{ap + b(1-p) \le y\} \\
&\qquad\qquad = P\{(a-b)p + b \le y\} \\
&\qquad\qquad = P\{p \le \frac{y-b}{a-b}\} \\
&\qquad = F_p(\frac{y-b}{a-b}) = \begin{cases} \frac{y-b}{a-b}, b \le y \le a \\ 0, y < b \\ 1, y > a \end{cases} \\
&f_Y(y) = F_Y(y)' = \begin{cases} \frac{1}{a-b}, b \le y \le a \\ 0, 其他 \end{cases} \quad \Rightarrow Y \sim U[b, a]
\end{aligned}
$$

可以看到，运算的结果服从以两亲代的基因值为端点的区间上的均匀分布。

另外，我们的亲代是一对坐标，因此需要两个随机权重。

交叉作为基因类的一个成员函数出现，具体实现的代码如下：

```
1  vector<Chromosome>
2      Chromosome::crossover(Chromosome & c1, Chromosome & c2) {
3      // A simulation of sigle-point-crossover.
4       uniform_real_distribution <double> distribution(0, 1);
5      double power_1 = distribution(_engine);
6      double power_2 = distribution(_engine);
7
8      Chromosome child_c1 = Chromosome(
9          c1._x1 * power_1 + c2._x1 * (1 - power_1),
10         c1._x2 * power_2 + c2._x2 * (1 - power_2));
11     Chromosome child_c2 = Chromosome(
12         c1._x1 * (1 - power_1) + c2._x1 * power_1,
13         c1._x2 * (1 - power_2) + c2._x2 * power_2);
14
```

```
15      return {child_c1, child_c2};
16  }
```

变异的策略 将影响基因变异的方式和概率，由于先前采用了实数编码的方式，变异的策略采用随机的方法来代替：

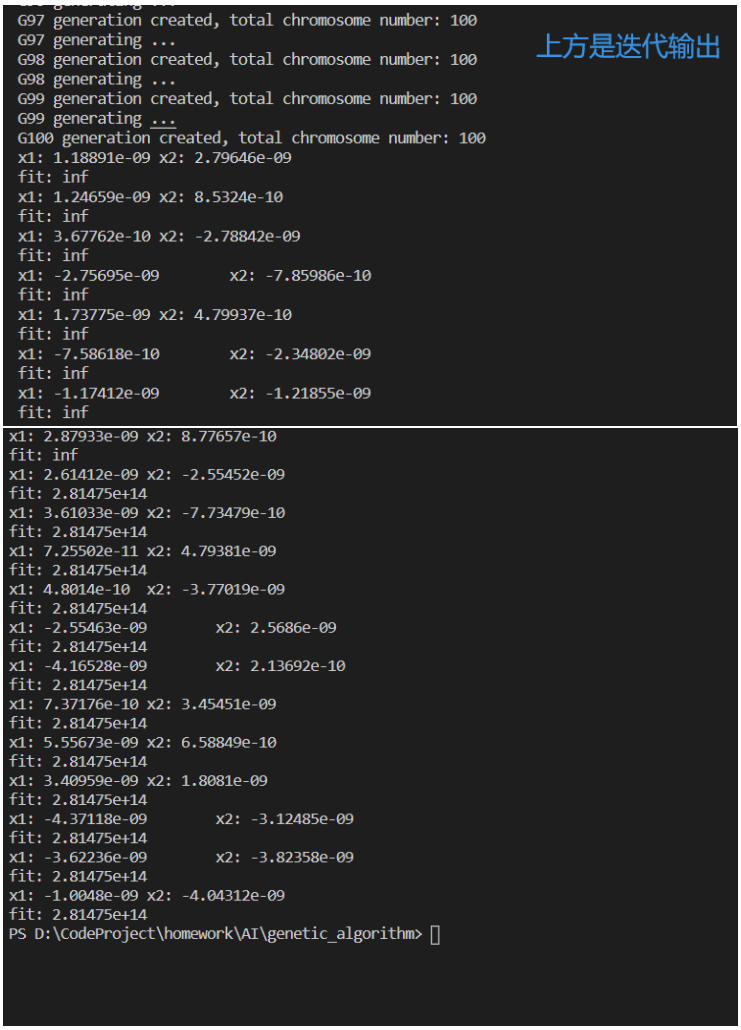
```
1  void Chromosome::random() {
2       uniform_real_distribution <double> distribution(-5, 5);
3      _x1 = distribution(_engine);
4      _x2 = distribution(_engine);
5  }
6
7  void Chromosome::mutate() {
8      random();
9  }
```

迭代和输出 建立在上述的各个过程的分析和实现的基础上，有了上述过程的基础，我们现在可以着手实现创建基因对象和迭代输出的实验了。

主函数中设定初始的群体状态：

```
1  int main {
2      Chromosomeset chromosome_set;
3      chromosome_set.create(20);
4      chromosome_set.generate(100);
5      chromosome_set.print();
6      return 0;
7  }
```

创建20个个体，进行100次迭代，并输出留存的基因信息。没有传参的print函数默认输出前20个基因个体。输出如下：



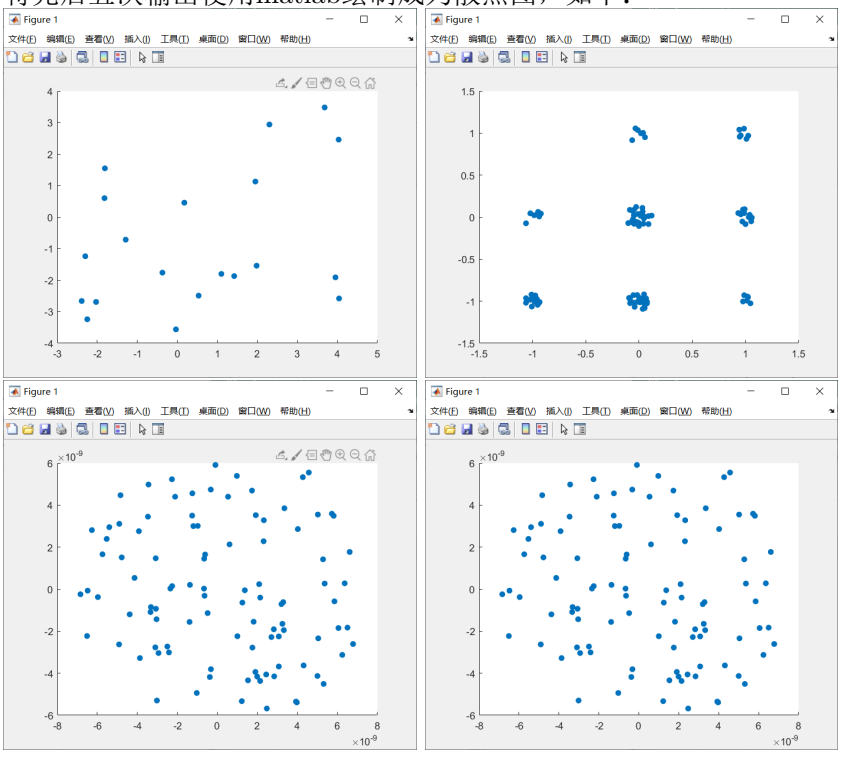
可以看到，适应度最高接近无穷，随着适应度的从低到高（输出是适应度从高到低），x1和x2的值都有向0收敛的趋势。

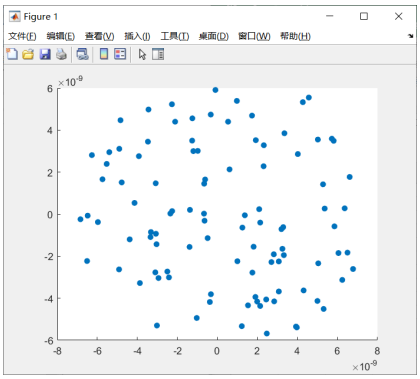
但，仅仅从终端界面的输出也许并不方便看出数据的分布情况，我们不妨使用matlab分阶段绘图。给源代码增添一些文件输出的方法之后，修改main函数代码如下：

```
1 int main {
2     Chromosomeset chromosome_set;
3     chromosome_set.create(20);
4     chromosome_set.write( "g0.txt" );
```

```
5
6       chromosome_set.generate(2);
7       chromosome_set.write("g2.txt");
8
9       chromosome_set.generate(8);
10      chromosome_set.write("g10.txt");
11
12      chromosome_set.generate(40);
13      chromosome_set.write("g50.txt");
14
15      chromosome_set.generate(100);
16      chromosome_set.write("g100.txt");
17  return 0;
18  }
```

将先后五次输出使用matlab绘制成为散点图，如下：





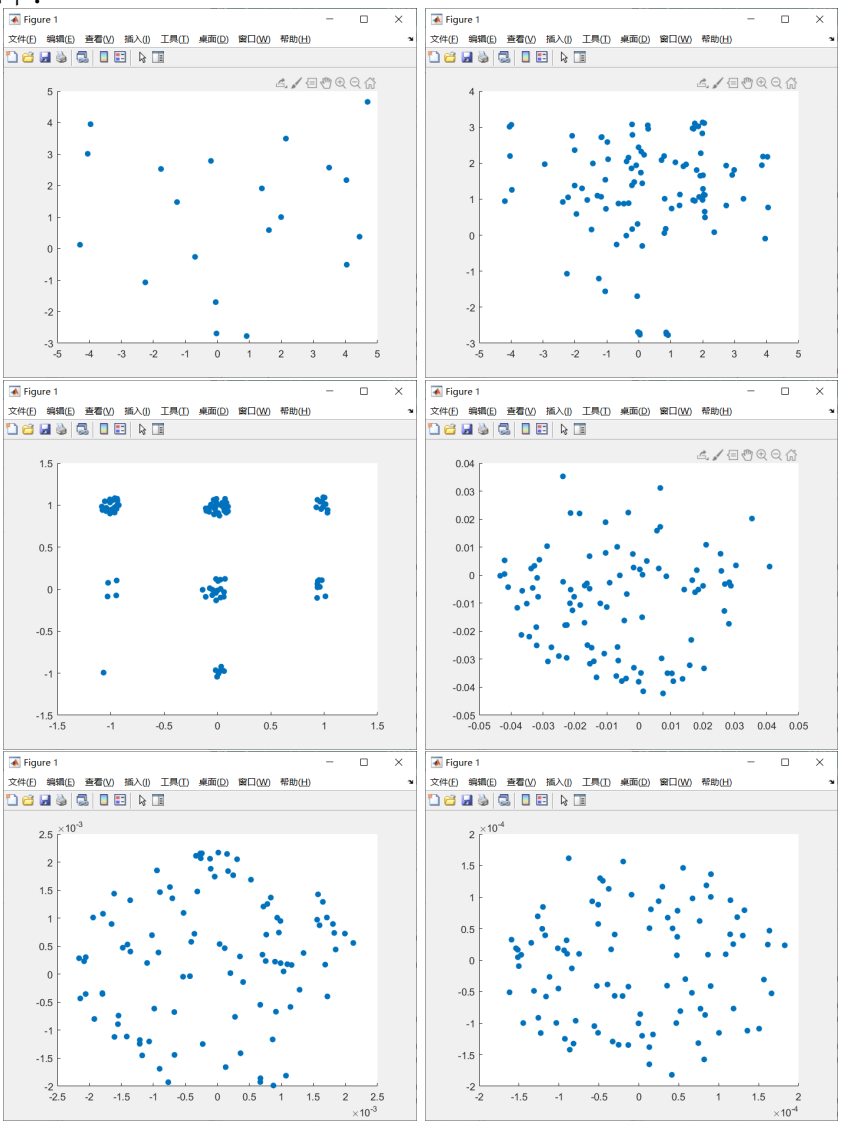
图片分别是迭代0次、2次、10次、50次、100次的输出。可以发现，前三张散点图反映了粗略的迭代过程，而迭代10次、50次、100次后输出的数据已经不再变化，因为解的精度已经超出C++数据类型double可以处理的范围。

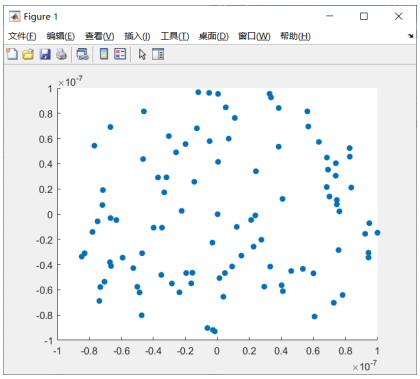
为了更好地观察迭代的过程，再进行一次实验，这一次降低迭代的次数，增加输出的频率：

```
1  int main () {
2      ChromosomeSet chromosome_set;
3      chromosome_set.create(20);
4      chromosome_set.write("g0.txt");
5
6      chromosome_set.generate(1);
7      chromosome_set.write("g1.txt");
8      chromosome_set.generate(1);
9      chromosome_set.write("g2.txt");
10     chromosome_set.generate(1);
11     chromosome_set.write("g3.txt");
12     chromosome_set.generate(1);
13     chromosome_set.write("g4.txt");
14     chromosome_set.generate(1);
15     chromosome_set.write("g5.txt");
16
17     chromosome_set.generate(5);
18     chromosome_set.write("g10.txt");
```

```
19
20      return 0;
21  }
```

将迭代0次、1次、2次、3次、4次、5次、10次的输出按顺序分别绘制输出如下：





第三次迭代输出后解聚集在几个局部最优解附近，而第四次迭代之后，所有局部最优解都被淘汰，并且从坐标轴刻度易知，种群不断向（0，0）点进化。

通过上面的迭代并观察结果，可以明显发现每迭代一次，解就在更靠近（0，0）点处聚集，可以推测（0，0）点大概率（在某个精度的误差允许下）是函数的最小值点。解有明显的向（0，0）点收敛的趋势。

4 总结

在实现算法时，应该先通过理论分析构造算法过程，再转化为代码，重点应该理解算法对遗传模拟的过程是如何在求解问题的过程中生效的。

使用面向对象的方法构建基因和基因集，将使代码构建的过程更加易于理解。本次算法中的变异操作直接使用了随机，也许并不是很好的解决方案，但最后的结果并没有受到影响。

通过对不同解的交叉、变异，生成一系列基因，并保留最接近函数最优解（最小值）的基因，也许可以认为这类似一个反向的拟合过程。

遗传算法最终的输出也许并不会是最优解，而是一系列接近最优解的基因的集合。

5 附录

基因集的迭代函数具体实现如下：

```
1 void ChromosomeSet::generate(int num) {
2     int generation = 0;
```

```
3
4       do {
5           cout << "G" << generation << " generating ..." << endl;
6
7           double total_fit = 0;
8           for (Chromosome & c : this->_set) {
9               double x1 = c.getX1();
10              double x2 = c.getX2();
11              double result = Function(x1, x2);
12
13              c.setFit(1 / result);
14              total_fit += 1 / result;
15          }
16
17          // Sort by fit descendingly.
18          sort(_set.begin(), _set.end(),
19              [](const Chromosome & c1, const Chromosome & c2) {
20                  return c1.getFit() > c2.getFit();
21              });
22
23
24          default_random_engine engine(time(nullptr));
25          uniform_real_distribution <double> distribution(0, 1);
26
27          vector<Chromosome> parent;
28          for (long long unsigned int i = 0; i < _set.size(); ++i) {
29              double rand_num = distribution(engine);
30              double add_up = 0;
31              for (Chromosome c : _set) {
32                  if (rand_num <= add_up) {
33                      parent.push_back(c);
34                  }
35                  else {
```

```
36                      add_up += c.getFit() /  total_fit ;
37                  }
38              }
39          }
40
41          if (parent.size() % 2 != 0 ) parent.pop_back();
42
43          for (auto iter = parent.begin(); iter < parent.end(); iter += 2) {
44              for (auto c : Chromosome::crossover(*iter, *(iter + 1))) {
45                  double rand_num = distribution(engine);
46                  if (rand_num <= 0.001) c.mutate();
47
48                  double x1 = c.getX1();
49                  double x2 = c.getX2();
50                  double result = Function(x1, x2);
51
52                  c.setFit(1 / result );
53
54                  _set.push_back(c);
55              }
56          }
57
58          // Sort by fit descendingly.
59          sort(_set.begin(), _set.end(),
60              [](const Chromosome & c1, const Chromosome & c2) {
61                  return c1.getFit() > c2.getFit();
62              });
63
64          while (_set.size() > _scale) {
65              _set.pop_back();
66          }
67
68
```

```
69          ++ generation;
70          cout << "G" << generation << " generation created, "
71              << "total chromosome number: " << _set.size() << endl;
72      } while (generation < num);
73  }
```

参考文献

[1] 李德毅,于剑,中国人工智能协会.人工智能导论[M].中国科学技术出版社:北京,2018:72.