

Санкт-Петербургский Национальный Исследовательский  
Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

**Лабораторная работа №2**  
**Создание генетического алгоритма средствами C#**

Выполнил  
Стафеев И.А.

Группа  
К3221

Проверил  
Иванов С.Е.

Санкт-Петербург,  
2025

## СОДЕРЖАНИЕ

Стр.

<b>ВВЕДЕНИЕ .....</b>	<b>3</b>
<b>1 Выполнение задания.....</b>	<b>4</b>
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>11</b>

## ВВЕДЕНИЕ

Цель работы: изучить генетический алгоритм и реализовать алгоритм средствами ООП для решения Диофантова уравнения.

Для достижения цели были поставлены следующие задачи:

1. Изучить общую схему устройства генетических алгоритмов;
2. Реализовать класс и присущие ему методы для "особи отождествляющей диофантово уравнение;
3. Реализовать схему генетического алгоритма с учетом особенностей разработанной "особи включая целевую функцию, кроссинговер и мутацию;
4. Посмотреть результат работы при различных параметрах алгоритма, таких как число начальных особей, доля выживших и так далее.

## 1 Выполнение задания

### Постановка задачи:

Генетический алгоритм основан на принципах естественного отбора и применяется для решения задач оптимизации. Решения представляются в виде генотипов — векторов генов, оцениваемых с помощью функции приспособленности. Алгоритм работает итерационно: на каждом шаге создается новое поколение с помощью операций кроссинговера, наследования, мутаций и селекции. Процесс продолжается до достижения оптимального решения или выполнения критерия останова, например, по числу поколений или времени работы.

В рамках этой лабораторной работы надо разработать генетический алгоритм, позволяющий решить диофантово уравнение  $a + 2b + 3c + 4d = 30$ ;

### Решение:

Вначале был создан вспомогательный класс **RandomOperations**, имеющий два метода: один для получения множества целых чисел из диапазона, и второй для получения подмножества целых чисел из диапазона. Эти методы будут применяться при выборе генов для мутации, а также для выбора, какие гены будут наследоваться от первого родителя, а какие - от второго. Код реализации этого класса показан на рисунке 1.

```
3  class RandomOperations
4  {
5      public static Random rnd = new Random();
6
7      public static HashSet<int> GetRandomFromRange(int min, int max, int count)
8      {
9          var numbers = new HashSet<int>();
10         while (numbers.Count < count)
11         {
12             numbers.Add(rnd.Next(min, max + 1));
13         }
14         return numbers;
15
16     public static HashSet<int> GetSetRange(int min, int max)
17     {
18         return new HashSet<int>(Enumerable.Range(min, max - min + 1));
19     }
20 }
21
```

Рисунок 1 — Класс RandomOperations

В качестве класса для "особи участвующей в эволюционном процессе, был создан класс **Eq**. Среди атрибутов класса - функция, отождествляющая уравнение, для которого ищется решение; требуемое значение функции; левая и правая граница допустимых значений параметров; максимальное число генов, которые могут мутировать и массив параметров. Поле **cur\_result** возвращает значение функции при подстановке начальных параметров, а поле **fitting\_f** - это целевая функция, в качестве которой была выбрана относительная ошибка от ожидаемого результата. Атрибуты и конструктор этого класса представлены на рисунке 2.

```

22  class Eq
23  {
24      public int[] nums;
25      public int expected_result;
26      public int left_border, right_border, num_of_mutations;
27      public Func<int[], int> f;
28      private static Random rnd = new Random();
29      public int cur_result
30      {
31          get { return f(nums); }
32      }
33      public double fitting_f
34      {
35          get { return (double)Math.Abs(cur_result - expected_result) / (double)Math.Abs(expected_result); }
36      }
37      public Eq(Func<int[], int> f, int expected_result, int left_border,
38              int right_border, int num_of_mutations, int[] nums)
39      {
40          this.f = f;
41          this.expected_result = expected_result;
42          this.nums = nums;
43          this.left_border = left_border;
44          this.right_border = right_border;
45          this.num_of_mutations = num_of_mutations;
46      }

```

Рисунок 2 — Атрибуты и конструктор класса Eq

Далее были созданы несколько методов, непосредственно вызываемых во время работы генетического алгоритма. Это метод **Mutate**, который изменяет несколько генов на случайное отклонение и метод **Print** для вывода генотипа (параметров) уравнения. Код приведен на рисунке 3.

```

47  public void Mutate()
48  {
49      var param_nums = RandomOperations.GetRandomFromRange(0, nums.Length - 1, num_of_mutations);
50      foreach (int par_num in param_nums)
51      {
52          int delta = rnd.Next(-2, 3);
53          nums[par_num] = Math.Clamp(nums[par_num] + delta, left_border, right_border);
54      }
55  }
56  Ссылка: 1
57  public void Print()
58  {
59      for (int i = 0; i < nums.Length; i++)
60      {
61          Console.Write("x{0}={1}, ", i + 1, nums[i]);
62      }
63      Console.WriteLine();

```

Рисунок 3 — Методы класса Eq

Также был создан оператор сложения, отвечающий за скрещивание двух особей. В результате применения оператора создается новая особь, которая получает часть признаков случайно от первого родителя и часть - от второго. Код методов и оператор приведены на рисунке 4.

```

64  Ссылка: 1
65  public static Eq operator +(Eq father, Eq mother)
66  {
67      int n = father.nums.Length;
68      var father_genes = RandomOperations.GetRandomFromRange(0, n - 1, n / 2);
69      var mother_genes = RandomOperations.GetSetRange(0, n - 1);
70      mother_genes.ExceptWith(father_genes);
71      int[] new_genes = new int[n];
72      foreach (int fgen in father_genes)
73      {
74          new_genes[fgen] = father.nums[fgen];
75      }
76      foreach (int mgen in mother_genes)
77      {
78          new_genes[mgen] = mother.nums[mgen];
79      }
80      return new Eq(father.f, father.expected_result, father.left_border, father.right_border,
81                  father.num_of_mutations, new_genes);
82  }

```

Рисунок 4 — Оператор сложения для класса Eq

Следом были реализованы статические методы класса **Eq** - для генерации особи со случайными параметрами при заданной функции уравнения и границах допустимых значений, а также для нахождения выборочного среднего и среднеквадратического отклонения для набора однотипных уравнений. Код этих методов показан на рисунке 5.

```

83 public static Eq GenerateEq(Func<int[], int> f, int expected_result, int left_border,
84                             int right_border, int num_of_mutations, int num_params)
85 {
86     // генерация уравнения со случайными параметрами при известном результате и функции
87     int[] nums = new int[num_params];
88     for (int i = 0; i < num_params; i++)
89     {
90         nums[i] = rnd.Next(left_border, right_border + 1);
91     }
92     return new Eq(f, expected_result, left_border, right_border, num_of_mutations, nums);
93 }
94
95 // Ссылка: 2
96 public static double FittingMean(List<Eq> eqs)
97 {
98     double sum = 0;
99     foreach (Eq eq in eqs)
100     {
101         sum += eq.fitting_f;
102     }
103     return sum / eqs.Count;
104 }
105
106 // Ссылка: 1
107 public static double GenStdev(List<Eq> eqs)
108 {
109     double mean = FittingMean(eqs);
110     double stdev = 0;
111     for (int i = 0; i < eqs.Count; i++)
112     {
113         stdev += Math.Pow(eqs[i].fitting_f - mean, 2);
114     }
115     return Math.Sqrt(stdev / eqs.Count);
116 }

```

Рисунок 5 — Статические методы класса Eq

В методе **Main** были определены начальные условия: функция диофантова уравнения, границы допустимых значений, доля выживающих особей и доля мутирующих. Далее создается выборка из случайных особей. Код показан на рисунке 6.

```

117 // Ссылка: 1
118 static void Main()
119 {
120     // параметры одного уравнения
121     Func<int[], int> f = (nums) => nums[0] + 2 * nums[1] + 3 * nums[2] + 4 * nums[3];
122     int expected_result = 30;
123     int left_border = 1, right_border = 30, num_of_mutations = 1;
124     // параметры поколения
125     int start_eq_num = 30;
126     double survive_ratio = 0.1;
127     double mutants_ratio = 0.1;
128     var rnd = RandomOperations.rnd;
129
130     var generations = new List<Eq>(start_eq_num);
131     for (int i = 0; i < start_eq_num; i++)
132     {
133         generations.Add(Eq.GenerateEq(f, expected_result, left_border, right_border, num_of_mutations, 4));
134     }
135 }

```

Рисунок 6 — Начальные условия в методе Main

На рисунке 7 показан сам генетический алгоритм. В бесконечном цикле особи сортируются для выбор лучших, причем в качестве ключа сортиров-

ки выступает сигма-отсечение  $p_i = 1 - \frac{f_i - \langle f \rangle}{2\sigma}$  ( $f$  - целевая функция), чтобы отсекал выбросные значения и алгоритм сходился медленнее, но без наличия "удобных" случайных особей. Если среди особей есть та, у которой значение целевой функции минимально и равно нулю, то это решение диофантова уравнения. Если таковой особи нет, берутся лучшие особи, а дефицит восполняется скрещиванием случайных особей (не обязательно лучших, чтобы не возник преобладающий генотип). В конце происходит случайная мутация.

```

134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
for (int i = 0; ; i++)
{
    var cur_mean = Eq.FittingMean(generations);
    var cur_stdev = Eq.GenStdev(generations);
    var best_gens = generations.OrderBy(g => 1 + (g.fitting_f - cur_mean) / (2 * cur_stdev)).ToList();
    var med = best_gens[start_eq_num / 2];

    Console.WriteLine("Generation {0}, FitMin {1}, FitMax {2}, FitMean {3}", i + 1, best_gens[0].fitting_f, best_

    if (best_gens.Any(g => g.fitting_f == 0))
    {
        best_gens[0].Print();
        break;
    }

    var new_generations = new List<Eq>(best_gens.Take((int)(survive_ratio * start_eq_num)));
    while (new_generations.Count < start_eq_num)
    {
        var father = generations[rnd.Next(start_eq_num)];
        var mother = generations[rnd.Next(start_eq_num)];
        var child = father + mother;
        new_generations.Add(child);
    }

    int mutants = (int)(mutants_ratio * start_eq_num);
    foreach (int eq_num in RandomOperations.GetRandomFromRange(0, start_eq_num - 1, mutants))
    {
        new_generations[eq_num].Mutate();
    }
    generations = new_generations;
}

```

Рисунок 7 — Реализация генетического алгоритма

Примеры работы генетического алгоритма при разных начальных параметрах приведены на рисунках 8, 9 и 10.



```
Консоль отладки Microsoft Visual Studio
Generation 8, FitMin 0,5333333333333333, FitMax 5,5, FitMean 2,6033333333333326
Generation 9, FitMin 0,3666666666666664, FitMax 5,6, FitMean 2,466666666666663
Generation 10, FitMin 0,3666666666666664, FitMax 4,566666666666666, FitMean 1,8900000000000001
Generation 11, FitMin 0,0666666666666667, FitMax 4,166666666666667, FitMean 1,68
Generation 12, FitMin 0,0666666666666667, FitMax 4,633333333333334, FitMean 1,693333333333331
Generation 13, FitMin 0,0666666666666667, FitMax 4,7, FitMean 1,518888888888889
Generation 14, FitMin 0,0666666666666667, FitMax 5,066666666666666, FitMean 1,686666666666667
Generation 15, FitMin 0,0666666666666667, FitMax 5,233333333333333, FitMean 1,4255555555555557
Generation 16, FitMin 0,0666666666666667, FitMax 4,1, FitMean 0,8177777777777778
Generation 17, FitMin 0,0666666666666667, FitMax 1,8, FitMean 0,5622222222222221
Generation 18, FitMin 0,0666666666666667, FitMax 1,3, FitMean 0,4733333333333333
Generation 19, FitMin 0,0666666666666667, FitMax 1,166666666666667, FitMean 0,3633333333333333
Generation 20, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,3044444444444435
Generation 21, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,2644444444444437
Generation 22, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,2533333333333324
Generation 23, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,2455555555555555
Generation 24, FitMin 0,0333333333333333, FitMax 0,8333333333333334, FitMean 0,2022222222222216
Generation 25, FitMin 0,0333333333333333, FitMax 0,7, FitMean 0,1755555555555555
Generation 26, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,1966666666666666
Generation 27, FitMin 0,0333333333333333, FitMax 0,966666666666667, FitMean 0,2355555555555552
Generation 28, FitMin 0,0333333333333333, FitMax 1,0333333333333334, FitMean 0,2399999999999988
Generation 29, FitMin 0,0333333333333333, FitMax 0,766666666666667, FitMean 0,2177777777777774
Generation 30, FitMin 0, FitMax 0,3666666666666664, FitMean 0,1255555555555555
x1=6, x2=4, x3=4, x4=1,
D:\ProgrammingProjects\itmo_OOP\sem2\labs\GeneticAlg\bin\Debug\net8.0\GeneticAlg.exe (процесс 6320) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 8 — Результат выполнения при 30 особях и 0.1 доле выживших

```
Консоль отладки Microsoft Visual Studio
Generation 1, FitMin 0,5333333333333333, FitMax 7,666666666666667, FitMean 4,306444444444443
Generation 2, FitMin 0,5333333333333333, FitMax 7,866666666666666, FitMean 4,347222222222224
Generation 3, FitMin 0,5, FitMax 8,166666666666666, FitMean 4,2737777777777755
Generation 4, FitMin 0,1666666666666666, FitMax 8,3, FitMean 4,217222222222221
Generation 5, FitMin 0,1666666666666666, FitMax 8,033333333333333, FitMean 4,27
Generation 6, FitMin 0,1, FitMax 8,033333333333333, FitMean 4,2090000000000005
Generation 7, FitMin 0,1, FitMax 7,833333333333333, FitMean 4,166777777777779
Generation 8, FitMin 0,1, FitMax 7,9, FitMean 4,041666666666665
Generation 9, FitMin 0, FitMax 7,733333333333333, FitMean 3,846111111111111
x1=3, x2=5, x3=3, x4=2,
D:\ProgrammingProjects\itmo_OOP\sem2\labs\GeneticAlg\bin\Debug\net8.0\GeneticAlg.exe (процесс 19484) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 9 — Результат выполнения при 300 особях и 0.01 доле выживших

```
Консоль отладки Microsoft Visual Studio
Generation 7, FitMin 0,2, FitMax 2,2, FitMean 1,243333333333332
Generation 8, FitMin 0,2, FitMax 2,2, FitMean 0,876666666666666
Generation 9, FitMin 0,2, FitMax 1,3, FitMean 0,563333333333332
Generation 10, FitMin 0,2, FitMax 1,366666666666667, FitMean 0,58
Generation 11, FitMin 0,2, FitMax 0,933333333333333, FitMean 0,386666666666666
Generation 12, FitMin 0,166666666666666, FitMax 0,866666666666667, FitMean 0,35
Generation 13, FitMin 0,2, FitMax 0,4, FitMean 0,273333333333333
Generation 14, FitMin 0,2, FitMax 0,333333333333333, FitMean 0,219999999999999
Generation 15, FitMin 0,133333333333333, FitMax 0,266666666666666, FitMean 0,196666666666666
Generation 16, FitMin 0,133333333333333, FitMax 0,266666666666666, FitMean 0,189999999999999
Generation 17, FitMin 0,133333333333333, FitMax 0,266666666666666, FitMean 0,183333333333333
Generation 18, FitMin 0,133333333333333, FitMax 0,266666666666666, FitMean 0,176666666666666
Generation 19, FitMin 0,1, FitMax 0,2, FitMean 0,153333333333332
Generation 20, FitMin 0,1, FitMax 0,266666666666666, FitMean 0,146666666666666
Generation 21, FitMin 0,066666666666667, FitMax 0,133333333333333, FitMean 0,113333333333333
Generation 22, FitMin 0,066666666666667, FitMax 0,133333333333333, FitMean 0,109999999999999
Generation 23, FitMin 0,066666666666667, FitMax 0,266666666666666, FitMean 0,116666666666667
Generation 24, FitMin 0,066666666666667, FitMax 0,233333333333334, FitMean 0,106666666666666
Generation 25, FitMin 0,066666666666667, FitMax 0,133333333333333, FitMean 0,093333333333332
Generation 26, FitMin 0,066666666666667, FitMax 0,1, FitMean 0,083333333333333
Generation 27, FitMin 0,066666666666667, FitMax 0,1, FitMean 0,073333333333333
Generation 28, FitMin 0,066666666666667, FitMax 0,133333333333333, FitMean 0,079999999999999
Generation 29, FitMin 0, FitMax 0,066666666666667, FitMean 0,06
x1=1, x2=11, x3=1, x4=1,
D:\ProgrammingProjects\itmo_OOP\sem2\labs\GeneticAlg\bin\Debug\net8.0\GeneticAlg.exe (процесс 8508) завершил работу с кодом 0 (0x0).
Чтобы автоматически закрывать консоль при остановке отладки, включите параметр "Сервис" ->"Параметры" ->"Отладка" -> "Автоматически закрыть консоль при остановке отладки".
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рисунок 10 — Результат выполнения при 10 особях и 0.5 доле выживших

## ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были выполнены все задачи, поставленная цель достигнута. Было изучено общее строение генетических алгоритмов, и на его основе был создан генетический алгоритм для нахождения решений диофантова уравнения. Полученные знания и навыки заложили основу понимания генетических алгоритмов и разработки и применения их в реальных промышленных задачах.