МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

НГТУ НЭТИ | **Факультет прикладной математики и информатики**

Кафедра прикладной математики

Практическая работа №3
по дисциплине «Цифровые модели и оценивание параметров»

Генетические алгоритмы

Студенты	БЕГИЧЕВ АЛЕКСАНДР
	ГРОСС АЛЕКСЕЙ
	ШИШКИН НИКИТА
Группа	ПМ-92
Преподаватели	ВАГИН Д.В.

Новосибирск, 2022

Цель работы

Реализовать решение обратной задачи с выбранным вариантом с помощью ПГА.

Вариант 5: С помощью ПГА восстановить коэффициенты полинома десятой степени по его значениям в заданном наборе точек. Значения зашумить на 1, 2, 5 %.

Теоретическая часть

Генетические алгоритмы – механизмы, основанные на принципах комбинаторного перебора вариантов решения оптимизационных задач в модели эволюции живых организмов на Земле.

Основные термины

1. Фенотип – строение любого живого организма, которое определяется наборов его генов, которые он получает от своих родителей.
2. Генотип – набор генов живого организма.
3. Мутации – изменения, которым подвергаются гены.
4. Приспособленность – вероятность выживаемости особи.

Концепция генетического алгоритма

Перенос природных сущностей и механизмов на решение обратных задач приводит к следующей концепции:

- искомые (оптимальные) параметры задают генотип с фенотипом, имеющим максимальную приспособленность (минимум функционала обратной задачи);
- процесс решения обратной задачи представляется в виде эволю- ции популяции особей, где каждое новое поколение имеет большую (среднюю) приспособленность, т. е. меньшее значение функционала обратной задачи;
- вероятность для конкретной особи того, что её гены будут пере- даны особям следующего поколения, зависит от её приспособленно- сти, что обеспечивает закрепление «хороших» генов в течение эволю- ционного процесса;
- при создании нового поколения с определённой вероятностью происходят мутации генов (изменения значений параметров), что потенциально обеспечивает нахождение минимально возможного при данной параметризации значения функционала обратной за- дачи.

Простой генетический алгоритм

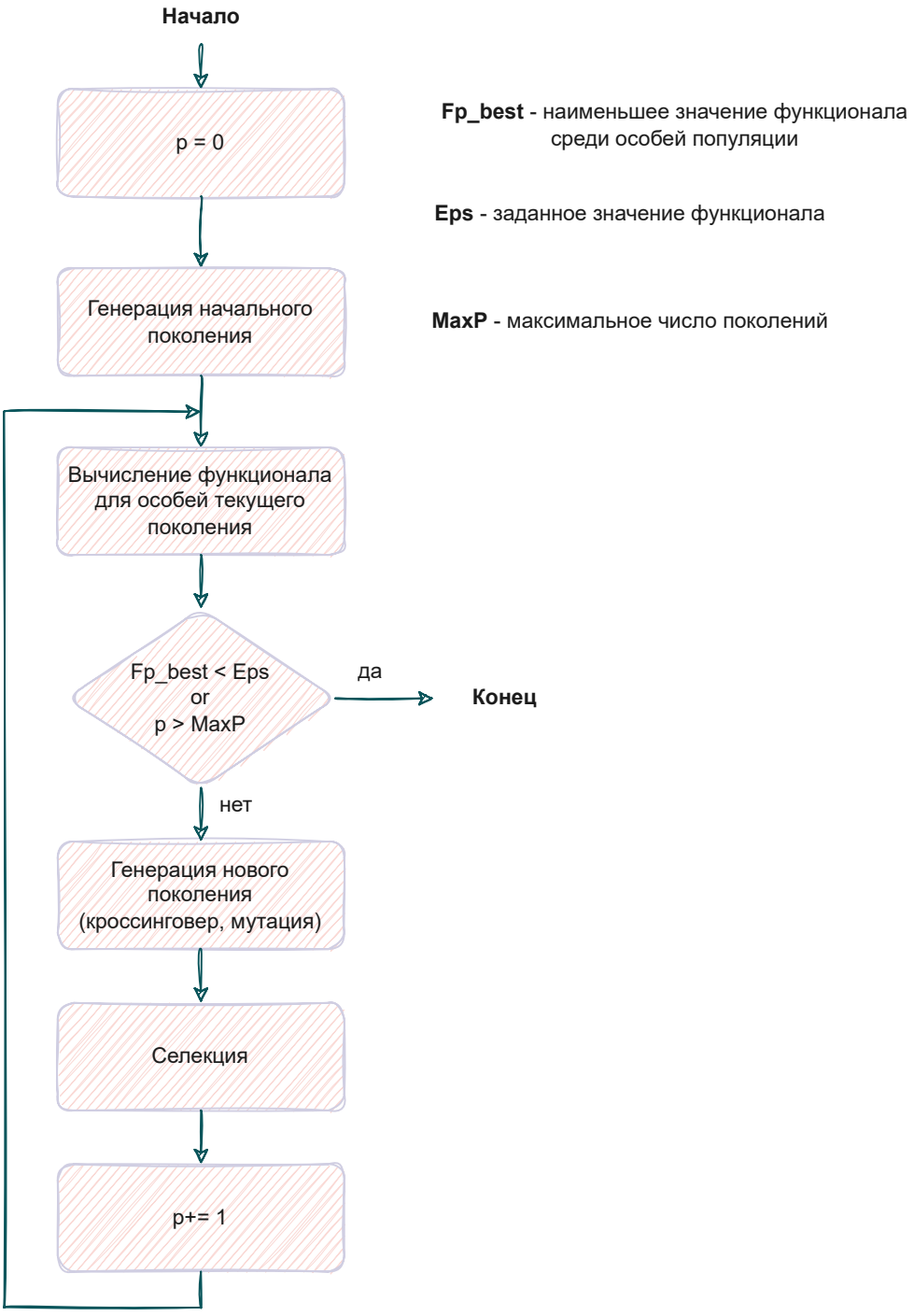

Рис. 1: Простой генетический алгоритм

Тестирование программы

Указанный в тесте начальный вектор фенотипа задается случайным образом. Фенотип новой особи вычисляется как значение полинома от x (каждого значения предыдущего фенотипа).

Тест 1

Параметры теста №1:

- фенотип = $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.5\}$;
- генотип = $\{1, 2, 3\}$.

Зашумление отсуствует.

Iteration	Functional
0	0.13
1	0.19
2	0.17
3	0.12
4	0.16
...	...
55	$4.09 \cdot 10^{-2}$
56	$2.42 \cdot 10^{-2}$
57	$3.06 \cdot 10^{-2}$
58	$3.04 \cdot 10^{-2}$
59	$2.86 \cdot 10^{-2}$
...	...
150	$1.77 \cdot 10^{-2}$
151	$1.29 \cdot 10^{-2}$
152	$4.6 \cdot 10^{-3}$
153	$9.19 \cdot 10^{-3}$
154	$1.38 \cdot 10^{-2}$
...	...
195	$1.02 \cdot 10^{-2}$
196	$3.7 \cdot 10^{-3}$
197	$6.25 \cdot 10^{-3}$
198	$4.6 \cdot 10^{-3}$
199	$5.45 \cdot 10^{-3}$

Genotype*
$9.998 \cdot 10^{-1}$
2.007208
2.993296

Тест 2

Параметры теста №2:

- фенотип = $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2, 3.5, 4.6, 5.5\}$;
- генотип = $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Зашумление отсуствует.

Iteration	Functional
0	16,646
1	1,573.56
2	1,199.74
3	1,208.15
4	1,027.1
...	...
55	639.68
56	642.35
57	643.26
58	643.25
59	641.27
...	...
150	630.56
151	634.59
152	635.49
153	640.3
154	634.94
...	...
195	631.94
196	639.62
197	639.97
198	633.16
199	635.43

Genotype*
0, 987087
2, 309579
0, 867729
8, 452442
5, 147074
8, 46360
0, 315069
0, 125526
5, 029269
8, 930723

Тест 3

Параметры теста №3:

- фенотип = $\{1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2, 3.5, 4.6, 5.5\}$;
- генотип = $\{1, 2, 3, 4, 5\}$.

Зашумление 5%.

Iteration	Functional
0	32.47
1	6.71
2	4.98
3	5.89
4	7.92
...	...
55	2.86
56	2.12
57	3.57
58	4.19
59	7.63
...	...
150	1.82
151	3.04
152	1.48
153	1.89
154	1.46
...	...
195	2.02
196	2.23
197	2.28
198	2.19
199	1.87

Genotype*
$0,984003$
$2,225736$
$1,793276$
$6,749756$
$2,972690$

Вывод

Из результатов тестирования видно, что при полиномах малой степени решение получается довольно точным. При использовании полиномов высокого порядка получит точное решение уже не удается. Это связано с выбором интервала для аргумента полинома, чем больше интервал, тем больший вклад при высоких степенях, которые перекрывают собой коэффициенты при малых степенях.

Также изменение функционала зависит от выбора алгоритма мутации. Если вероятность мутаций будет чрезмерно высока, то решение задачи может стать проблематичным, так как мутация может разрушить генотипы с высокой приспособленностью. Полное отсутствие мутаций также недопустимо, так как в этом случае найти оптимум может стать в принципе невозможно, если, например, гены с «правильными» значениями вообще не присутствуют в исходной популяции.

Листинг

```
var phenotype = new List<double> { 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2 };
var genotype = new List<double> { 1, 2 };

SimpleGeneticAlgorithm sga = new(genotype, phenotype, 1);
sga.Inverse();

foreach(var item in sga.Population[0].Genotype)
{
    Console.WriteLine(item);
}
```

```
namespace problem_3;

public class SimpleGeneticAlgorithm
{
    private const int PopulationSize = 10000;
    private const int MaxParent = 10;
    private const double MinFunctional = 1E-7;
    private const double MutationProbability = 0.99;
    private readonly IList<double> _genotype;
    private readonly IList<double> _phenotype;
    private readonly double _noise;
    private readonly Specimen _realSpecimen;
    private IList<Specimen> _population;
    private IList<Specimen> _newPopulation;

    public ImmutableArray<Specimen> Population => _population.ToImmutableArray();

    public SimpleGeneticAlgorithm(IList<double> genotype, IList<double> phenotype,
    ↪  double noise)
    {
        _genotype = genotype;
        _phenotype = phenotype;
        _noise = noise;

        _realSpecimen = new(genotype);
        _realSpecimen.SetPhenotype(phenotype);

        _population = new List<Specimen>();
        _newPopulation = new List<Specimen>();
    }

    private void PrimaryPopulation()
    {
        for (int i = 0; i < PopulationSize; i++)
        {
            List<double> specimenGenotype = new();

            for (int j = 0; j < _genotype.Count; j++)
            {
                specimenGenotype.Add(new Random().NextDouble() * _phenotype.Count);
            }

            _population.Add(new Specimen(specimenGenotype));
            _population[i].SetPhenotype(_phenotype);
            _population[i].SetFunctional(_realSpecimen);
```

```
        }

        _population = new List<Specimen>(_population.OrderBy(specimen =>
↪   specimen.Functional));
    }

    private Specimen Child(Specimen father, Specimen mother)
    {
        List<double> genotype = new();

        int icrossingover = new Random().Next(0, _genotype.Count);

        for (int i = 0; i < icrossingover; i++)
        {
            genotype.Add(father.Genotype[i]);
        }

        for (int i = icrossingover; i < _genotype.Count; i++)
        {
            genotype.Add(mother.Genotype[i]);
        }

        return new(genotype);
    }

    private void Mutation(Specimen specimen)
    {
        double prob = new Random().NextDouble();

        if (prob < MutationProbability)
        {
            int igen = new Random().Next(0, _genotype.Count);
            specimen.Mutation(igen, new Random().NextDouble() * _phenotype.Count);
        }
    }

    private void NewPopulation()
    {
        _newPopulation = new List<Specimen>(_population);

        for (int i = 0; i < MaxParent; i++)
        {
            for (int j = 0; j < PopulationSize / MaxParent; j++)
            {
                // Произвольный номер второго родителя
                int k = new Random().Next(0, PopulationSize);

                var child = Child(_population[i], _population[k]);

                Mutation(child);
                _newPopulation.Add(child);
            }
        }

        for (int i = 0; i < PopulationSize * 2; i++)
        {
            _newPopulation[i].SetPhenotype(_phenotype);
            _newPopulation[i].SetFunctional(_realSpecimen);
        }

```

```
        _newPopulation = _newPopulation.OrderBy(specimen =>
↪   specimen.Functional).ToList();
    }

    private double Selection()
    {
        var functional = 1e+30;

        for (int i = 0; i < PopulationSize; i++)
        {
            _population[i] = _newPopulation[i];
        }

        if (_population[0].Functional < functional)
        {
            functional = _population[0].Functional;
        }

        return functional;
    }

    public void Inverse()
    {
        NoisyValues();
        PrimaryPopulation();

        var functional = 1e+30;

        if (_population[0].Functional < functional)
        {
            functional = _population[0].Functional;
        }

        Console.WriteLine($"{0}:\tfunctional = {functional}");

        for (int iter = 1; iter < 200; iter++)
        {
            NewPopulation();
            functional = Selection();

            Console.WriteLine($"{iter}:\tfunctional = {functional}");

            if (functional < MinFunctional) break;
        }
    }

    private void NoisyValues()
    {
        for (int i = 0; i < _genotype.Count; i++)
        {
            _genotype[i] += _genotype[i] * (_noise / 10.0);
        }
    }
}
```

```
namespace problem_3;

public class Specimen
{
    private readonly IList<double> _genotype;
    private List<double>? _phenotype;
    public double Functional { get; private set; }
    public ImmutableArray<double> Genotype => _genotype.ToImmutableArray();

    public Specimen(IList<double> genes) => _genotype = genes;

    private double PolynomValue(double x)
    {
        double value = 0.0;
        double factor = 1.0;

        // Начиная со старшей степени
        for (int i = _genotype.Count - 1; i >= 0; i--)
        {
            value += factor * _genotype[i];
            factor *= x;
        }

        return value;
    }

    public void SetPhenotype(IList<double> phenotype)
    {
        _phenotype ??= new(new double[phenotype.Count]);

        for (int i = 0; i < phenotype.Count; i++)
        {
            _phenotype[i] = PolynomValue(phenotype[i]);
        }
    }

    public void SetFunctional(Specimen realSpecimen)
    {
        for (int i = 0; i < _phenotype!.Count; i++)
        {
            Functional += Math.Abs(_phenotype[i] - realSpecimen._phenotype![i]);
        }
    }

    public void Mutation(int igen, double mutation)
        => _genotype[igen] = mutation;
}
```