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

Группа ПМ-92 АРТЮХОВ РОМАН

ВАСЬКИН ЛЕОНИД

Преподаватели ВАГИН ДЕНИС ВЛАДИМИРОВИЧ

Новосибирск, 2022

Цель работы:

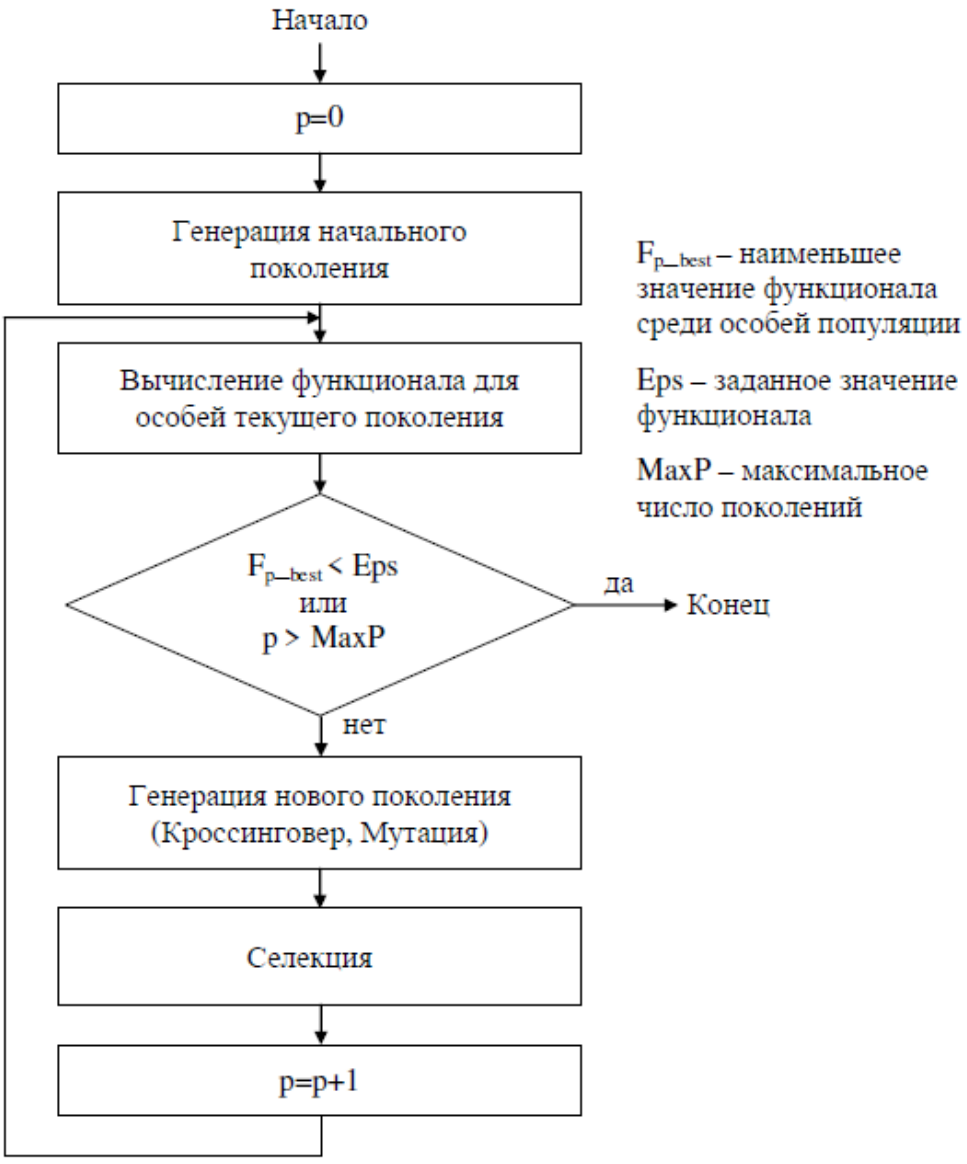
Реализовать ПГА для решения обратных задач.

Задание (вариант 2):

Реализовать ПГА. Генотип состоит из N чисел. Фенотип состоит из M чисел. Числа фенотипа вычисляются регулярным образом из чисел генотипа.

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

ПГА:



- **Входные данные**

```
//* Данные для задачи
public double[] Coefficients      { get; set; }   /// Коэффициенты полинома (генотип)
public double[] Points            { get; set; }   /// Заданный набор точек (фенотип)
public double MinFunctional       { get; set; }   /// Значение фукнционала для выхода
public double MutationProbability { get; set; }   /// Вероятность мутации
public uint CountGen              { get; set; }   /// Число генов
public uint CountPopulation       { get; set; }   /// Число особей в популяции
public uint CountGeneration       { get; set; }   /// Число генераций
public uint MaxParent             { get; set; }   /// Максимальное число родителей
```

- **Генерация начального поколения**

Создание особей. Для каждой особи рандомно выбираются гены, высчитывается фенотип особи по созданному генотипу. По созданному фенотипу высчитывается функционал особи. Далее особь добавляется в поколение.

```
// Генерация начального поколения
Population = new List<Being>();
for (int i = 0; i < CountPopulation; i++) {
    Vector<double> Gens = new Vector<double>((int)CountGen);
    for (int j = 0; j < CountGen; j++)
        Gens[j] = (GetRandomDouble(MinValue, MaxValue));

    var being = new Being(Gens);
    being.SetPhenotip(Points);
    being.Functional = Functional(being, TrueBeing);
    Population.Add(being);
}
Population = Population.OrderBy(item => item.Functional).ToList();
```

- **Высчитывание функционала особи**

Значение функционала – сумма модулей разности чисел фенотипа текущей особи и фенотипа истинной особи.

```
//* Расчет функционала
public static double Functional(Being CurBeing, Being TrueBeing) {
    double f = 0;
    for (int i = 0; i < CurBeing.Phenotype.Length; i++)
        f += Abs(TrueBeing.Phenotype[i] - CurBeing.Phenotype[i]);
    return f;
}
```

- **Кроссинговер**

На вход приходят 2 родителя. Генерируются две случайные точки. До первой точки, гены берутся от отца. От первой точки до второй гены матери. От второй точки до конца, снова берутся гены отца.

```
//* Кроссинговер (скрещивание отца и матери)
public Being GetChild(Being mother, Being father) {
    Being child = new Being((int)CountGen, Points.Length);
    int index1 = (new Random()).Next(0,      (int)CountGen); // Первая точка кроссинговера
    int index2 = (new Random()).Next(index1, (int)CountGen); // Вторая точка Кроссинговера

    // Первая часть генотипа от отца
    for (int i = 0; i < index1; i++)
        child.Genotype[i] = father.Genotype[i];

    // Вторая часть генотипа от матери
    for (int i = index1; i < index2; i++)
        child.Genotype[i] = mother.Genotype[i];

    // Третья часть генотипа от отца
    for (int i = index2; i < CountGen; i++)
        child.Genotype[i] = father.Genotype[i];

    return child;
}
```

- **Мутация**

Генерируется вероятность мутации. Если она меньше входной мутации, мутируем особь. Выбираем ген для мутации и задаем для этого гена случайное число.

```
//* Проявление мутации у существа
public void Mutation(Being being) {
    double probability = GetRandomDouble(0, 1);  // Вероятность мутации

    // Если меньше заданной мутации (мутируем существо)
    if (probability < MutationProbability) {
        int index = (new Random()).Next(0, (int)CountGen); // Рандомное место для мутации
        being.Genotype[index] = GetRandomDouble(MinValue, MaxValue);
    }
}
```

- **Селекция**

Сортируем новое поколение по значению функционала. Первую половину поколения отбираем, вторую уничтожаем.

```
//* Селекция (отбор лучших существ)
public void Selection(List<Being> population, double bestFunctional) {
    // Сортируем новое поколение
    Population = population.OrderBy(item => item.Functional).ToList();

    // Оставляем лучших существ
    Population.RemoveRange((int)CountPopulation, (int)CountPopulation);
}
```

Исследования:

- **Размер фенотипа (5, 10, 20 точек)**

$$\begin{cases} Coefficients : [1,2,3] \\ Points : [0.1,0.2,0.3,0.4,0.5] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [0,568, 2,311, 2,953]; Functional = 0,03791658068773085

$$\begin{cases} Coefficients : [1,2,3] \\ Points : [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [1,407, 1,493, 3,132]; Functional = 0,273717064297085

$$\begin{cases} Coefficients : [1, 2, 3] \\ Points : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [1,288, 1,354, 3,294]; Functional = 1,4645878006383968

Чем меньше размер фенотипа, функционал ниже, но плохая точность решения. Чем больше размер фенотипа, функционал выше, но получше точность решения.

- **Размер генотипа (3, 6, 10)**

$$\begin{cases} Coefficients : [1, 2, 3] \\ Points : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [0,800, 2,222, 2,951]; Functional = 0,12100022993219817

$$\begin{cases} Coefficients : [1, 2, 3, 4, 5, 6] \\ Points : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 6 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод:

Gens: [0,863, 2,210, 4,573, 0,902, 6,671, 5,780]; Functional = 0,3008514004074039

$$\begin{cases} Coefficients : [1,2,3,4,5,6,7,8,9,10] \\ Points : [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.8 \\ CountGen : 10 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод:

Gens: [2,288, 1,311, 3,105, 1,389, 5,835, 6,641, 8,810, 6,343, 9,276, 9,996];

Functional = 0,1219316917995652

- **Вероятность мутации (0.2)**

$$\begin{cases} Coefficients : [1,2,3] \\ Points : [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.2 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [1,691, 1,096, 3,253]; Functional = 0,4932298154722412

- **Вероятность мутации (0.99)**

$$\begin{cases} Coefficients : [1,2,3] \\ Points : [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1] \\ MinFunctional : 0.01 \\ MutationProbability : 0.99 \\ CountGen : 3 \\ CountPopulation : 40 \\ CountGeneration : 1000 \\ MaxParent : 4 \end{cases}$$

Вывод: Gens: [1,161, 1,813, 3,044]; Functional = 0,1017336436644447

Мутация очень хорошо влияет как на функционал, так и на получаемое решение.