Реализация просто генетического алгоритма

Красноярск 2020

**СОДЕРЖАНИЕ**

[1. Общие сведения 5](#_Toc38238401)

[1.1. Основные понятия 5](#_Toc38238402)

[1.2. Понятие эволюционной стратегии 5](#_Toc38238403)

[1.3. Достоинства эволюционного алгоритма 5](#_Toc38238404)

[1.4. Недостатки эволюционного алгоритма 6](#_Toc38238405)

[2. Описание системы разработанной эволюционной стратегии 7](#_Toc38238406)

[2.1. Требования к системе 7](#_Toc38238407)

[2.2. Описание используемых технологий 8](#_Toc38238408)

[2.3. Описание архитектуры системы 8](#_Toc38238409)

[2.4. Описание класса Program 10](#_Toc38238410)

[2.5. Описание класса Population 11](#_Toc38238411)

[2.6. Описание класса Individ 14](#_Toc38238412)

[2.7. Интерфейс системы 18](#_Toc38238413)

[ЗАКЛЮЧЕНИЕ 21](#_Toc38238414)

[СПИСОК СОКРАЩЕНИЙ 22](#_Toc38238415)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc38238416)

ВВЕДЕНИЕ

Эволюционные алгоритмы – интеллектуальная система принятия решений, основная идея алгоритма которой основывается на принципах действия эволюционных стратегий. Эволюционны алгоритмы решают широкий спектр задач, моделирование, автоматическое управление, задачи планирования, задачи проектирования, численная комбинаторика и др. Эволюционные алгоритмы представляет из себя имитацию развития и эволюцию различных организмов и популяций в мире биологии. Все эксперименты по эволюционным алгоритмам проводятся в искусственной среде, которая имитирует окружающую среду для индивидов в цифровом формате. Окружающая среда может являться просто набором наилучших параметров индивида, например целевой функцией, к которой будет стремиться развитие популяции, или быть набором элементов, с которым особи будут взаимодействовать, и из-за этого будут изменяться их параметры.

Существует несколько видов эволюционных алгоритмов, у каждого есть свои особенности, свои достоинства и недостатки, но все они решают поставленные задачи схожим алгоритмом:

1. Эволюционные стратегии. Особь или индивид в этом виде эволюционных алгоритмов представлен в удобном для компьютера виде, в виде вещественных чисел. Основная операция, которая используется в эволюционных стратегиях это – мутация, операторы и основные процессы в эволюционных алгоритмах я опишу ниже. Так же вспомогательной операцией для развития популяции в этом виде эволюционных алгоритмов является скрещивание и дисперсия.
2. Эволюционное программирование. Этот тип эволюционных алгоритмов можно характеризовать тем, что генотип индивида является программой, которая перезаписывается в ходе изменения популяции. Наиболее приспособленные программы к окружающим условиям проходят на этап селекции. В эволюционном программировании применяются операторы скрещивания и мутации.
3. Генетические алгоритмы. В этом виде генетических алгоритмов генотип особи(индивида) представлен в виде бинарной строки. Основные операторы, позволяющие изменять параметры особей из популяции в популяции это скрещивание, как и в живой природе, но вспомогательным оператором является мутация.
4. Генетическое программирование. Тип реализации эволюционных алгоритмов, при котором основным оператором является мутация. Последовательность генотипа особи так же, как и в генетических алгоритмах, описанных выше, представлен в виде бинарной строки.
5. Классифицирующие системы. Основной целью этого вида эволюционных алгоритмов является способность отличить один набор признаков особи от набора признаков другой особи, чтобы параметры имели наилучший параметр пригодности.
6. Адаптивный случайный поиск. Отличительная особенность этого типа эволюционных алгоритмов в том, что в ходе работы поиска программа находит все лучшие параметры для настройки внутренних систем поиска и обработки информации.

# **Общие сведения**

* 1. Основные понятия

Изменение качеств особей или индивидов определяется с помощью нескольких простых процессов: **селекции, скрещивания и мутации**.

*Селекция* представляет из себя выбор наиболее приспособленных к выживанию в условиях окружающей среды индивидов. Для определения индивидов для селекции необходимо определить *функцию пригодности,* которая будет выдавать значение параметра, и при определенных значениях параметра, особь будет пригодна для дальнейшей селекции.

*Скрещивание* предполагает смешивание “генов” ­– различных значений параметров особей, которые были выбраны на этапе селекции. Смешивание генов может происходить разными способами. Скрещивание должно нести случайный характер, но в некоторых алгоритмах его упрощают, определяют точки скрещивания, и т.д.

*Мутация* определяет случайное изменение генотипа особи с определенной вероятностью, при бинарном представлении генотипа мутация изменяет бит на противоположное значение, при вещественном представлении изменяет значение числа в положительную или отрицательную сторону на несколько значений.

*Мутанты* ­– особи, к которым был применен оператор мутации.

*Генотип* – набор параметров индивида, аналог хромосомы, может быт представлен в виде бинарной строки или вещественных чисел.

*Фенотип* – описание особи(индивида) в том виде, в котором он существует.

* 1. Понятие эволюционной стратегии

Эволюционная стратегия – эвристический метод оптимизации, основанный на эволюции и адаптации. Основным отличием является то, что генотип особей представлен в виде ряда вещественных чисел, который может определять фенотип особи или другие физические качества, которые влияют на вероятность выживания в окружающей среде. Как я уже упоминал ранее, основной оператор, за счет которого достигается конечная цель, является мутация, вспомогательным же оператором является скрещивание.

* 1. Достоинства эволюционного алгоритма

К достоинствам эволюционных алгоритмов можно отнести:

* 1. Не требуют дополнительной информации о поверхности ответа;
  2. Прекрасно работают при решении задач многоцелевой оптимизации;
  3. Просты и прозрачны в реализации;
  4. Решения легко поддаются интерпретации;
  5. Могут быть использованы в задачах с изменяющейся средой.
  6. Недостатки эволюционного алгоритма

Недостатки эволюционных алгоритмов:

1. Нет никакой гарантии о получении оптимального решения за конечное время;
2. Слабое теоретическое обоснование (слабая сходимость по вероятности);
3. Высокие вычислительные затраты;
4. Конфигурация крайне непростая (кодирование решения).
5. Описание системы разработанной эволюционной стратегии
   1. Требования к системе
6. **Тип представления**

Система должна иметь вещественное представление генотипа особи и целевой функции.

1. **Способ кодирования и декодирования индивидов**

При вещественном кодировании преобразование индивида в генотип не требуется, так как генотип и фенотип тождественны.

1. **Количество индивидов**

Количество индивидов в популяции можно выбирать самим. Но число должно быть кратно 4, потому что при рекомбинации особей используется турнирная сетка, включающая в себя 4 лучших представителя популяции, при скрещивании которые производят по 25% от общей суммы особей в популяции.

1. **Функция пригодности**

Функция пригодности для селекции будет представлять из себя сумму разностей генотипов особи и целевой функции. Целевая функция как набор параметров каждый раз будет определяться случайным способом.

1. **Селекция**

Селекция в данной системе должна представлять селекцию по турнирной сетке. Отбираются 4 особи и скрещиваются между собой определенным способом, чтобы произвести потомство, размером в 25% от общей суммы особей в популяции.

1. **Рекомбинация**

Механизм скрещивания будет включать в себя случайную рекомбинацию генов родителей, то есть на каждый ген потомка может попасть один из генов родителей в случайном порядке.

1. **Мутация**

Мутация должна быть основана на входном параметре вероятности мутации. Этот параметр определяет вероятность мутации особи и вероятность мутации каждого гена этой особи. Так как система представляет генотип особи в вещественном виде, то необходимо ввести параметр, который определит максимальное значение, на которое сможет мутировать один ген, при этом, мутация гена может происходить как в положительную, так и в отрицательную сторону.

1. **Условие останова**

Условием останова является достижение целевой функции хотя бы одной особью. То есть, наличие в популяции хотя бы одного индивида, у которого параметр пригодности будет равен нулю, генотип особи не будет отличаться от целевой функции.

1. **Проблема недопустимых индивидов**

Решение проблемы недопустимых индивидов будет происходить при помощи метода «Смертельный штраф», в ходе которого они будут просто отбрасываться. Этот метод является самым простым и самым действенным.

* 1. Описание используемых технологий

Для разработки собственной системы эволюционной стратегии я использовал объектно-ориентированный, строго типизированный язык программирования платформы Microsoft .NET Framework – C#. В ходе разработки я не использовал сторонние инструменты, модули или библиотеки, которые реализуют функционал эволюционных алгоритмов. Всю архитектуру и логику взаимодействия системы я разрабатывал сам, “с нуля”, имея только теоретические знания о работе эволюционных алгоритмов.

Для вывода информации использовал консольное приложение. Для реализации математических операций типа взятия числа по модулю и т.д, я использовал служебную библиотеку с набором математических функций Math.

Весь код содержит комментарии, необходимые для более глубокого понимая функционала того или иного участка кода.

* 1. Описание архитектуры системы

Структура системы состоит из 3-х больших классов, два из них являются функциональными, один доменным. Класс Program является шаблонным при создании консольного приложения, он является точкой входа и открытия программы. Класс Population определяет все функции и процессы, связанные с группой индивидов и их развитием, например, итерация в жизненном цикле, или стартовая итерация. Класс Individ (Individual) является доменным и содержит поля и функции определяющие процессы, связанные с деятельностью одной особи или индивида, так же содержит в себе генотип и параметр пригодности особи.

Диаграмму классов для разработанной системы вы можете наблюдать ниже:

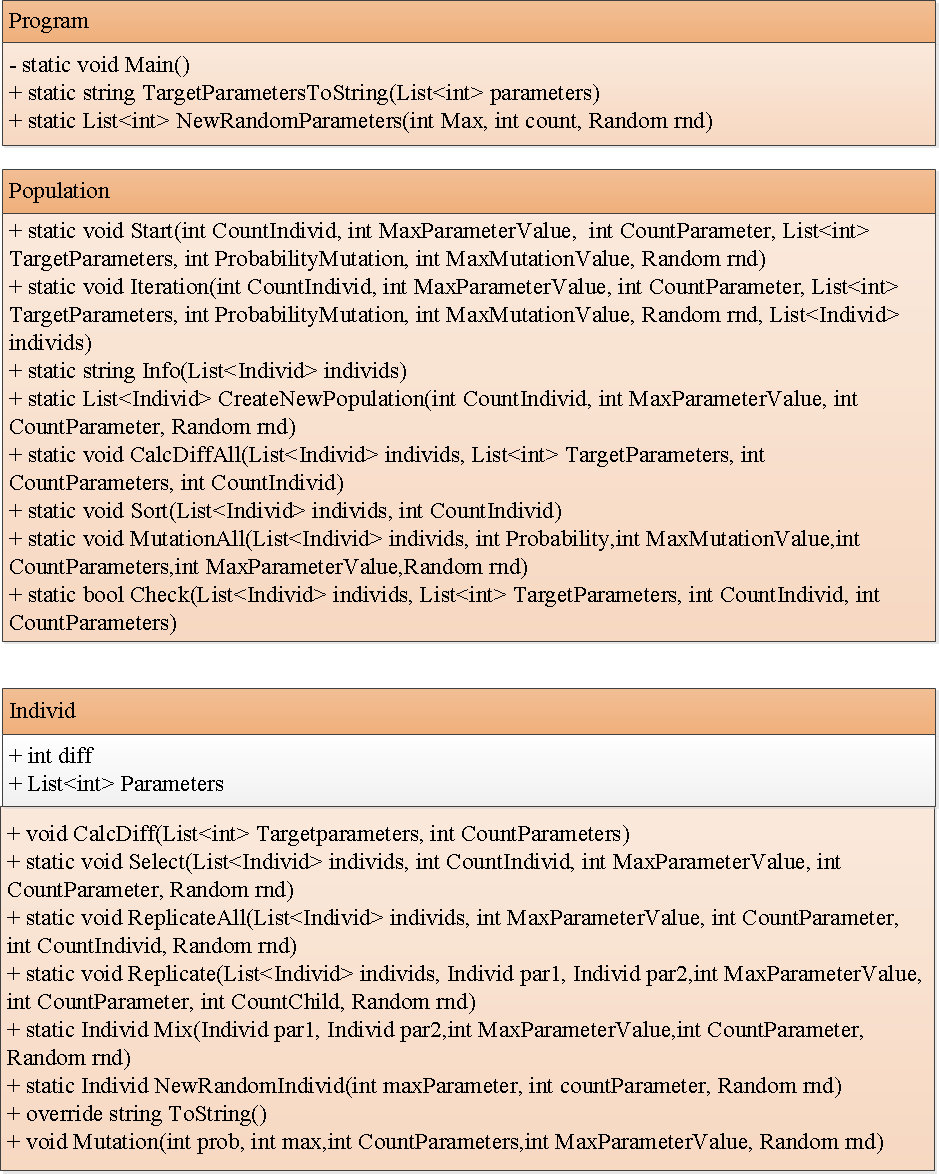


Рисунок 1 – Диаграмма классов.

На диаграмме классов «–» обозначает приватную функцию или переменную, а «+» публичную. «Static» означает статичное поле или функцию, которая выполняется без необходимости наличия экземпляра класса, то есть при прямом обращении к классу. «Void» имеет значение процедуры, которая не возвращает какие-либо значения. «Override» означает переопределяемую стандартную функцию.

* 1. Описание класса Program

Главный метод «Main» является точкой входа в приложение.

static void Main()

{

//Количество особей, кратное 4

int CountIndivid = 100;

//Максимально значение параметра

int MaxParameterValue = 10;

//Количество параметров

int CountParameter = 10;

//Вероятность мутации, %

int ProbabilityMatation = 80;

//Максимально значение мутации

int MaxMutationValue = 2;

//Случайная величина

Random rnd = new Random();

//Лист с параметрами, к которым стремятся индивиды, заполняется случайно

List<int> TargetParameters = NewRandomParameters(MaxParameterValue, CountParameter ,rnd);

//Старт ГА

Population.Start(CountIndivid, MaxParameterValue, CountParameter, TargetParameters, ProbabilityMatation, MaxMutationValue, rnd);

//Ожидание завершения работы кода

Console.ReadKey();

}

Этот метод содержит ряд параметров, которые можно изменять для достижения различного результата. В эти параметры входит:

* Количество особей
* Максимально значение гена (т.к. ген представлен в виде вещественного числа)
* Количество генов
* Вероятность мутации особи и гена
* Максимально значение, на которое может мутировать ген

Для различных моментов в системе, когда требуется событие со случайной вероятность тут же, в этом методе создаем переменную «rnd», которая будет отвечать за генерацию случайных чисел.

Так же, здесь указана целевая функция, это набор генов, к которому стремятся все особи в популяции как самому оптимальному генотипу. Эволюционная стратегия считается выполненной если хоть одна особь в популяции достигла такого набора генов, что является точкой останова.

Этот метод содержит вызов процедуры класса «Population» который начинает весь алгоритм – «Start».

public static string TargetParametersToString(List<int> parameters)

{

string ret = "Искомые параметры:\n";

foreach (var a in parameters)

{

ret = ret + Convert.ToString(a)+" ";

}

ret += "\n\n";

return ret;

}

Класс «Program» так же содержит служебную функцию «TargetParametersToString», которая позволяет выводит целевую функцию при каждой итерации на экран в виде одной строки.

//Метод заполнения параметров случайными значениями

public static List<int> NewRandomParameters(int Max, int count, Random rnd)

{

List<int> list = new List<int>();

for (int i = 0; i < count; i++)

{

list.Add(rnd.Next(Max));

}

return list;

}

Метод «NewRandomParameters» принимает входные параметры, такие как максимально значение гена, количество генов и переменную случайной генерации чисел. Эта функция возвращает лист с новыми параметрами в виде вещественных чисел.

* 1. Описание класса Population

Этот функциональный класс содержит методы и процедуры, которые необходимы для функциональности группы особей как популяции.

//Главный метод ГА

public static void Start(

int CountIndivid,

int MaxParameterValue,

int CountParameter,

List<int> TargetParameters,

int ProbabilityMutation,

int MaxMutationValue,

Random rnd)

{

//Создание новой популяции

List<Individ> individs = Population.CreateNewPopulation(CountIndivid,MaxParameterValue,CountParameter,rnd);

int NumberIteration = 0;

bool end = false;

do

{

//Вывод искомых параметров

Console.WriteLine(Program.TargetParametersToString(TargetParameters));

Console.WriteLine("Итерация номер - " + NumberIteration+"\n");

end = Check(individs, TargetParameters, CountIndivid, CountParameter);

Iteration(CountIndivid,

MaxParameterValue,

CountParameter,

TargetParameters,

ProbabilityMutation,

MaxMutationValue,

rnd,

individs);

NumberIteration++;

}

while (end == false);

}

Главная, начальная функция этого класса «Start» начинает весь алгоритм эволюционной стратегии. Принимает все параметры, передаваемые из класса «Program». Реализует цикл, из которого и вызываются все итерации и проверки на условия точки останова.

//Метод одной итерации ГА

public static void Iteration(

int CountIndivid,

int MaxParameterValue,

int CountParameter,

List<int> TargetParameters,

int ProbabilityMutation,

int MaxMutationValue,

Random rnd,

List<Individ> individs)

{

//Селекция по пригодности(остается 4 лучших особи)

Individ.Select(individs, CountIndivid, MaxParameterValue, CountIndivid, rnd);

//Скрещивание

//Выбираю случайные гены от родителей и вставляю потомкам

//смешиваю 1 и 2, 2 и 4, 1 и 3, 2 и 4 родителя, каждая пара дает 1/4 всей популяции

Individ.ReplicateAll(individs, MaxParameterValue, CountParameter, CountIndivid, rnd);

//Мутация

//Вероятность мутации индивида и гена одинакова

//При захождении мутации за край значения параметра она возвращается с другой стороны, значения циклично.

MutationAll(individs, ProbabilityMutation, MaxMutationValue, CountParameter, MaxParameterValue, rnd);

}

Эта процедура одной итерации, сама по себе не несет никаких преобразований, но вызывает другие необходимые функции и процедуры.

//Метод вывода информации о поколении

public static string Info(List<Individ> individs)

{

string ret = "";

foreach (var a in individs)

{

ret+=a.ToString()+"\n";

}

ret += "\n\n";

return ret;

}

Функция «Info» необходима для вывода информации о популяции, которая включает в себя отсортированных по параметру пригодности, описание генотипа каждой особи в популяции.

//Метод создания новой популяции

public static List<Individ> CreateNewPopulation(int CountIndivid, int MaxParameterValue, int CountParameter, Random rnd)

{

//Создание листа с индивидами

List<Individ> individs = new List<Individ>();

for (int i = 0; i < CountIndivid; i++)

{

individs.Add(Individ.NewRandomIndivid(MaxParameterValue, CountParameter, rnd));

}

return individs;

}

Данный метод необходим для создания новой популяции при старте эволюционной стратегии. Возвращает лист из экземпляров класса «Individ».

//Метод подсчета пригодности всей популяции

public static void CalcDiffAll(List<Individ> individs, List<int> TargetParameters, int CountParameters, int CountIndivid)

{

for (int i = 0; i < CountIndivid; i++)

{

individs[i].CalcDiff(TargetParameters,CountParameters);

}

}

Вышеописанный метод необходим для расчёта параметра пригодности для всех особей в популяции.

//Метод сортировки по пригодности (чем меньше, тем лучше)

public static void Sort(List<Individ> individs, int CountIndivid)

{

Individ temp;

for (int x = 0; x < CountIndivid; x++)

{

for (int y = 0; y < CountIndivid; y++)

{

if (individs[x].diff < individs[y].diff)

{

temp = individs[x];

individs[x] = individs[y];

individs[y] = temp;

}

}

}

}

Данный метод необходим для сортировки особей в популяции по параметру пригодности. Чем ниже параметр пригодности, тем лучше особь приспособлена к выживанию, тем выше ее место в рейтинге.

//Метод мутации всех особей

public static void MutationAll(List<Individ> individs, int Probability,int MaxMutationValue,int CountParameters,int MaxParameterValue,Random rnd)

{

foreach(var a in individs)

{

a.Mutation(Probability, MaxMutationValue, CountParameters, MaxParameterValue, rnd);

}

}

Данный метод позволяет провести мутацию у всех особей репутации.

//Проверка на выполнение задачи

public static bool Check(List<Individ> individs, List<int> TargetParameters, int CountIndivid, int CountParameters)

{

//Подсчет пригодности всей популяции

CalcDiffAll(individs, TargetParameters, CountParameters, CountIndivid);

//Сортировка по пригодности(чем меньше значение, тем лучше и выше место в списке)

Sort(individs, CountIndivid);

//Вывод информации о популяции

Console.WriteLine(Info(individs));

if (individs[0].diff == 0)

return true;

else

return false;

}

Данный метод позволяет проверить популяцию на условия точки останова. Возвращает значение типа bool.

* 1. Описание класса Individ

Класс Individ (Individual) является доменным и содержит поля и функции определяющие процессы, связанные с деятельностью одной особи или индивида, так же содержит в себе генотип и параметр пригодности особи.

//Значение пригодности(чем оно меньше, тем больше совпадает с искомым, тем лучше)

public int diff;

Данная переменная необходима для хранения значения пригодности особи.

//Метод вычисления пригодности

public void CalcDiff(List<int> Targetparameters, int CountParameters)

{

this.diff = 0;

for(int i =0;i<CountParameters;i++)

{

if(this.Parameters[i]> Targetparameters[i])

{

diff += this.Parameters[i] - Targetparameters[i];

}

else

{

diff += Targetparameters[i] - this.Parameters[i];

}

}

}

Вышеописанная процедура необходима для вычисления значения пригодности конкретной особи.

//Метод селекции

public static void Select(List<Individ> individs, int CountIndivid, int MaxParameterValue, int CountParameter, Random rnd)

{

Individ par1 = individs[0];

Individ par2 = individs[1];

Individ par3 = individs[2];

Individ par4 = individs[3];

individs.RemoveAll(x => x is Individ);

individs.Add(par1);

individs.Add(par2);

individs.Add(par3);

individs.Add(par4);

}

Данная процедура необходима для реализации функционала селекции, отбора наиболее приспособленных особей.

//Метод скрещивания всех родителей

public static void ReplicateAll(List<Individ> individs, int MaxParameterValue, int CountParameter, int CountIndivid, Random rnd)

{

Individ.Replicate(individs, individs[0], individs[1], MaxParameterValue, CountParameter, CountIndivid / 4, rnd);

Individ.Replicate(individs, individs[2], individs[3], MaxParameterValue, CountParameter, CountIndivid / 4, rnd);

Individ.Replicate(individs, individs[0], individs[2], MaxParameterValue, CountParameter, CountIndivid / 4, rnd);

Individ.Replicate(individs, individs[1], individs[3], MaxParameterValue, CountParameter, CountIndivid / 4, rnd);

individs.Remove(individs[0]);

individs.Remove(individs[0]);

individs.Remove(individs[0]);

individs.Remove(individs[0]);

}

Данная процедура необходима для осуществления функционала рекомбинации отобранных особей.

//Метод скрещивания двух индивидов

public static void Replicate(List<Individ> individs, Individ par1, Individ par2,int MaxParameterValue, int CountParameter, int CountChild, Random rnd)

{

for(int i =0;i<CountChild;i++)

{

individs.Add(Individ.Mix(par1, par2,MaxParameterValue,CountParameter,rnd));

}

}

Данный блок кода необходим для реализации скрещивания двух особей.

//Метод смешения генов

public static Individ Mix(Individ par1, Individ par2,int MaxParameterValue,int CountParameter, Random rnd)

{

Individ NewInd=Individ.NewRandomIndivid(MaxParameterValue,CountParameter,rnd);

for(int i = 0; i<CountParameter;i++)

{

if(rnd.Next(2)==0)

{

NewInd.Parameters[i] = par1.Parameters[i];

}

else

{

NewInd.Parameters[i] = par2.Parameters[i];

}

}

return NewInd;

}

Данный код необходим для создания генотипа потомка из генотипа двух особей предков.

//Метод создания нового случайного индивида

public static Individ NewRandomIndivid(int maxParameter, int countParameter, Random rnd)

{

Individ ind = new Individ();

ind.Parameters = Program.NewRandomParameters(maxParameter, countParameter, rnd);

return ind;

}

Эта функция необходима для создания нового индивида со случайными параметрами при стартовой инициализации популяции.

//Лист с параметрами

List<int> Parameters;

Данная переменная содержит в себе ряд параметров, которые определяют генотип особи.

//Метод для вывода информации

public override string ToString()

{

string ret = "";

foreach(var a in this.Parameters)

{

if(a==10)

{

}

ret = ret + Convert.ToString(a)+" ";

}

ret += " " + this.diff;

return ret;

}

Вышеописанный блок кода содержит переопределяемую функцию, которая служит для удобного вывода информации о генотипе особи и ее параметре пригодности.

//Метод мутации

public void Mutation(int prob, int max,int CountParameters,int MaxParameterValue, Random rnd)

{

if(max>=MaxParameterValue)

{

max = MaxParameterValue - max;

}

//Мутация индивида

if(rnd.Next(100)<prob)

{

for(int i=0;i<CountParameters;i++)

{

//Мутация гена

if(rnd.Next(100) < prob)

{

//Мутация в одну сторону

if(rnd.Next(2)==0)

{

this.Parameters[i] += rnd.Next(max);

if(this.Parameters[i]>=MaxParameterValue)

{

this.Parameters[i] = this.Parameters[i] - MaxParameterValue;

}

}

//в другую

else

{

this.Parameters[i] -= rnd.Next(max);

if (this.Parameters[i] < 0)

{

this.Parameters[i] = MaxParameterValue - Math.Abs(this.Parameters[i]);

}

}

}

}

}

}

Данный участок кода позволяет особи мутировать.

* 1. Интерфейс системы

В требованиях к разработке системы нет описания необходимости менять параметры при выполнении программы, поэтому я решил использовать интерфейс типа консоли. Данный интерфейс прост в реализации и не требует большой вычислительной мощности во время работы.



Рисунок 2 – Интерфейс системы.

На рис. 2 изображен интерфейс системы, вверху перед каждой итерацией выводится целевая функция – случайный набор параметров. Матрица, занимающая большую часть интерфейса это генотип особей. Справа находится параметр пригодности каждой особи, чем он ниже, тем лучше особь приспособлен к окружающей среде, и тем выше ее место в рейтинге. Как можно заметить особи отсортированы по этому параметру.

# **ЗАКЛЮЧЕНИЕ**

Мною была разработана данная система. Архитектура данной системы разделена на функциональные и доменные классы. Весь код системы был прокомментирован для дальнейшей возможной работы с ним.

Мною был составлен отчет, в котором описывается краткие сведения о эволюционных алгоритмах и стратегиях, об основных терминах и понятиях, использующихся при работе с данным видам систем искусственного интеллекта. В отчете мной был предоставлен разбор всех функций и процедур, использующихся в системе.

# **СПИСОК СОКРАЩЕНИЙ**

ЭА – эволюционный алгоритм

ЭС – эволюционная стратегия

ПО – программное обеспечение

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Пятаева, А. В. Интеллектуальные системы и технологии: учеб. пособие /А. В. Пятаева, К. В. Равич. – Красноярск: Сиб. федер. ун-т, 2018. –144 с.
2. Пенькова, Т. Г. Модели и методы искусственного интеллекта: учеб. пособие /Т. Г. Пенькова, Ю. В. Вайнштейн. – Красноярск: Сиб. федер. ун-т, 2019. – 116 с.
3. Круглова Т.Н., Власов А.С.: Методы искусственного интеллекта в мехатронике и робототехнике. 2017.
4. Панченко Т.В.: Генетические алгоритмы. 2007.
5. Рутковская Д., Пилиньский М., Рутковский Л.: Нейронные сети, генетические алгоритмы и нечеткие системы. 2006.