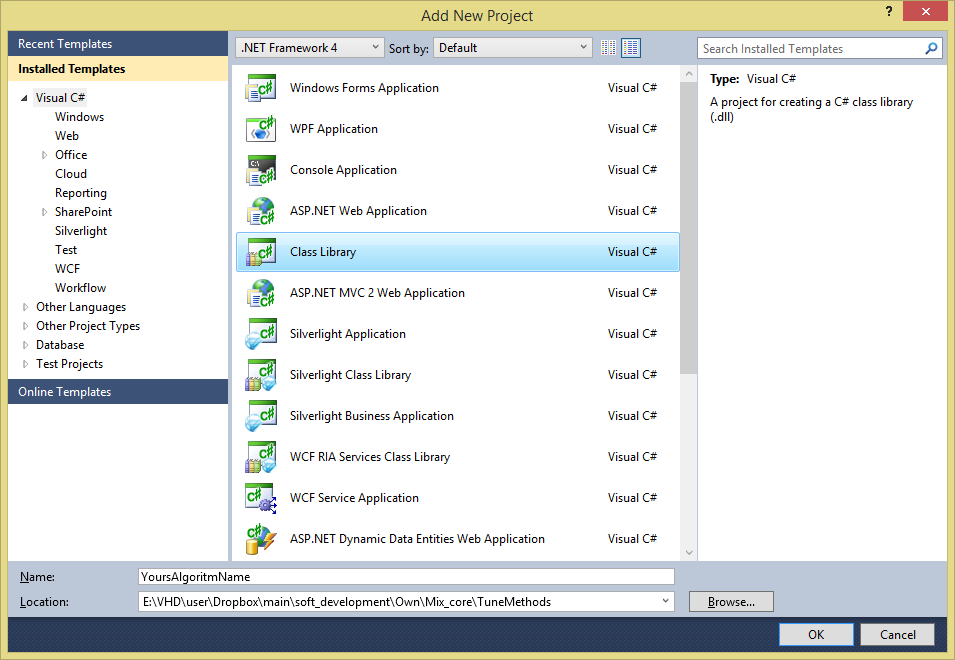
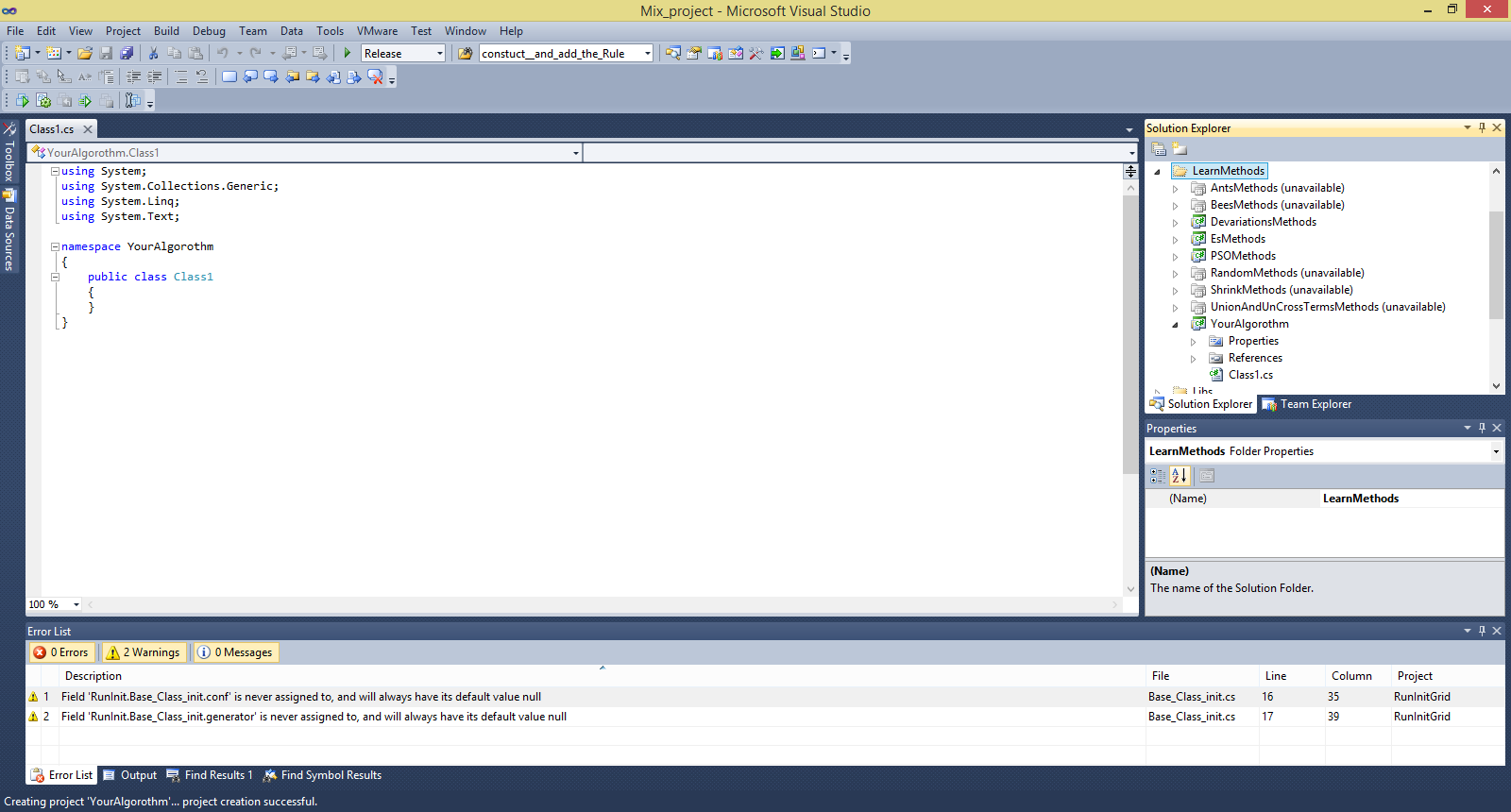
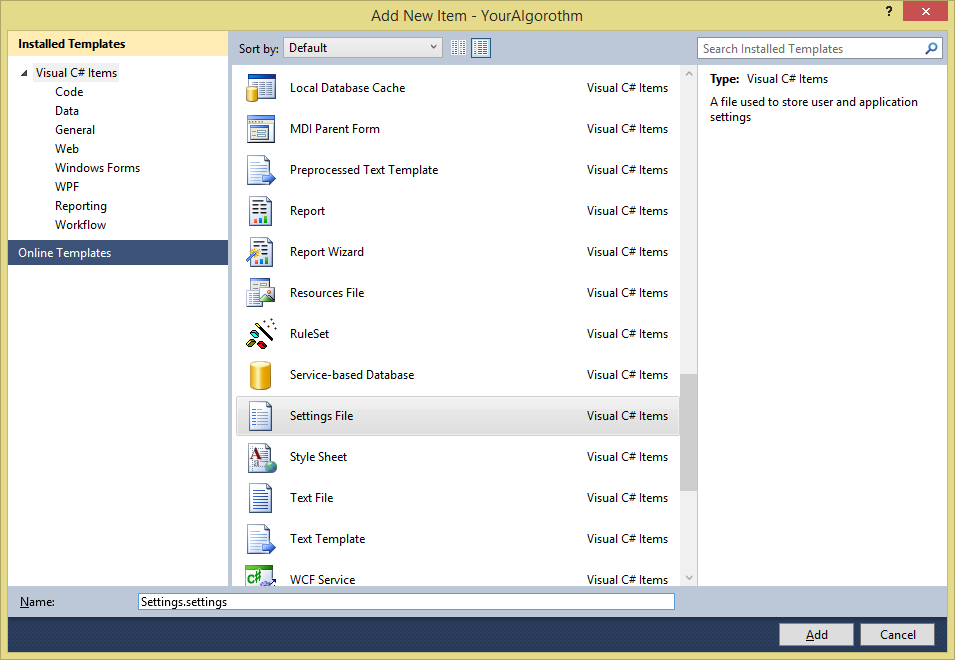
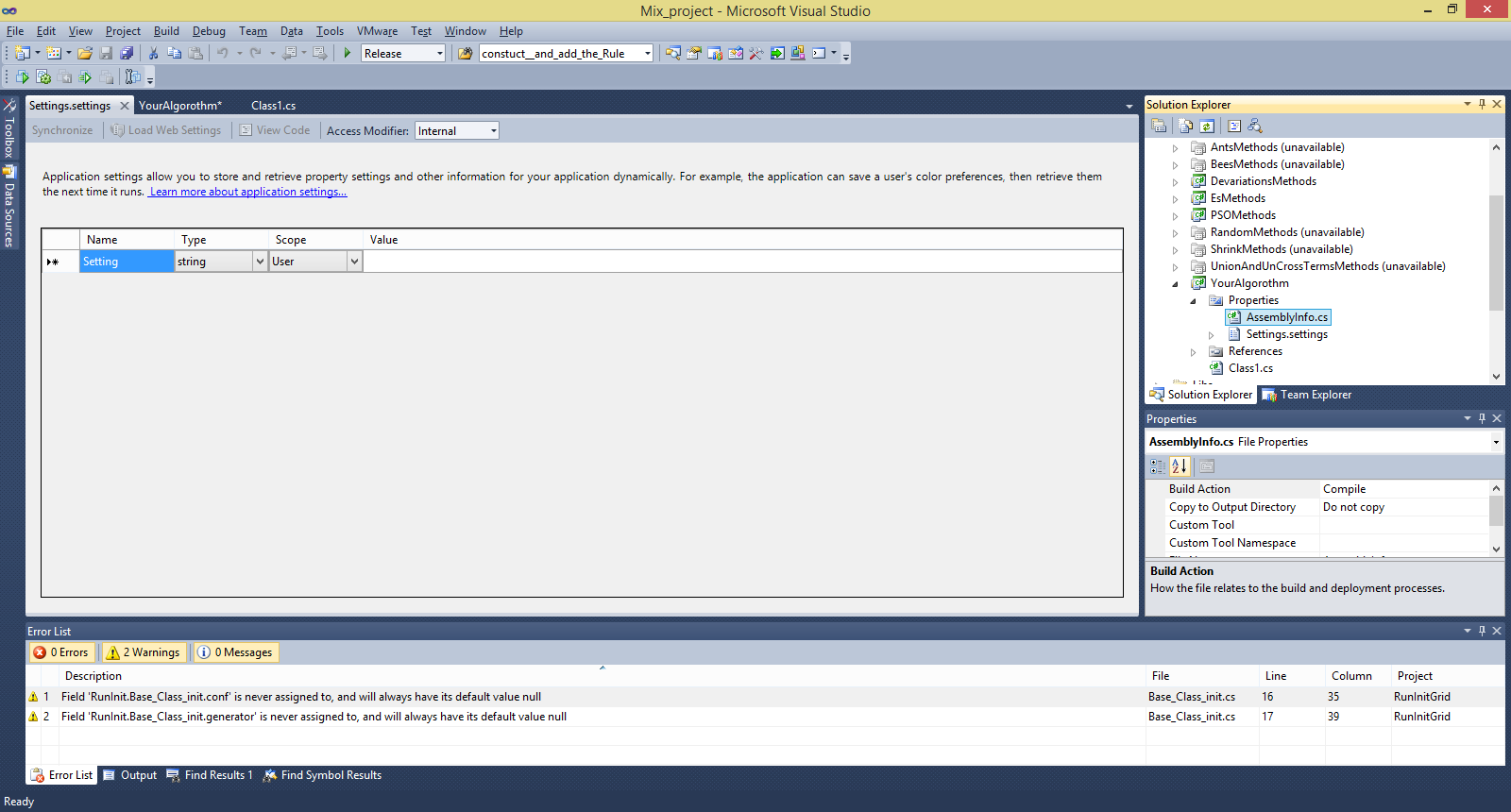
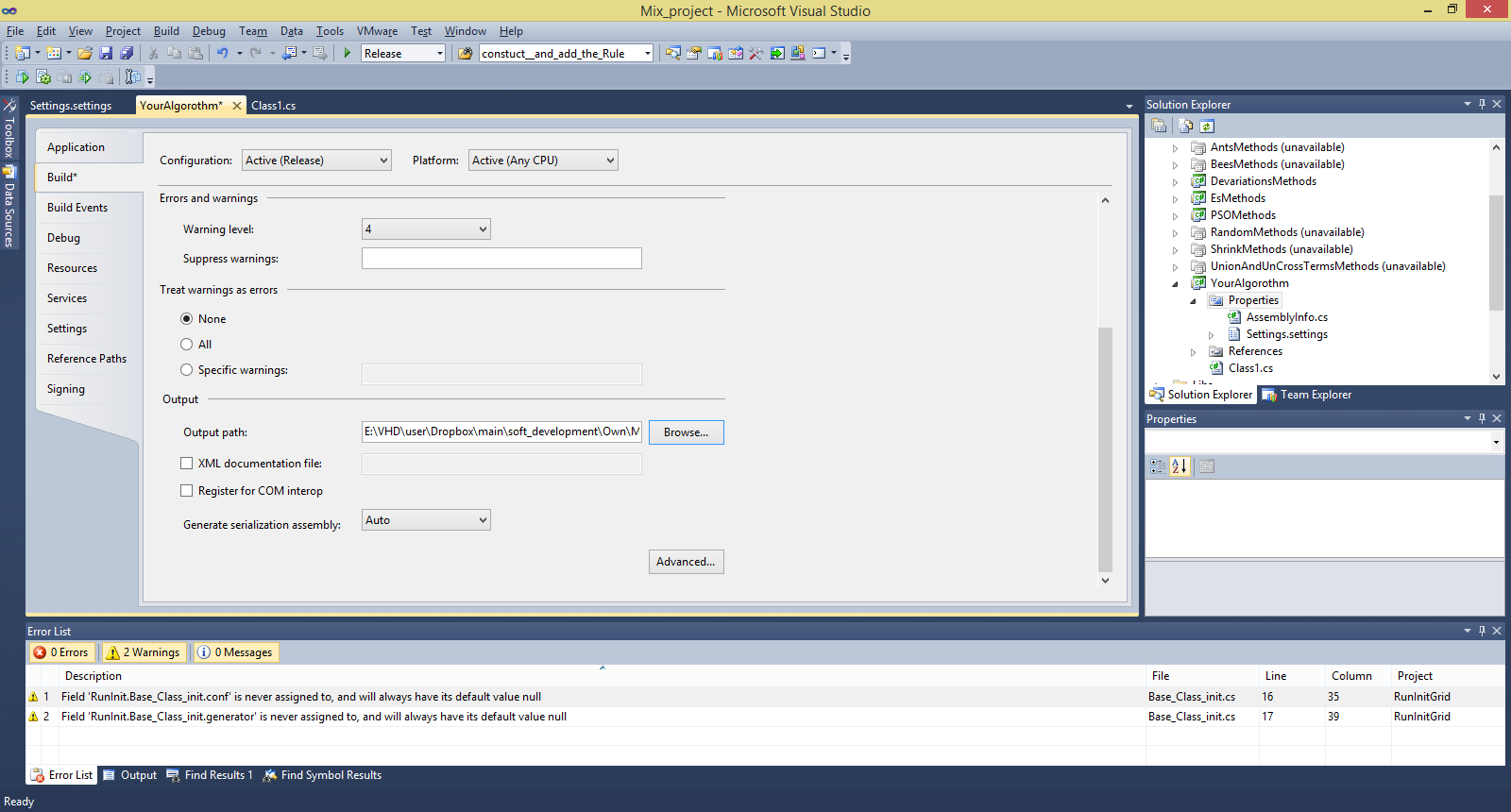
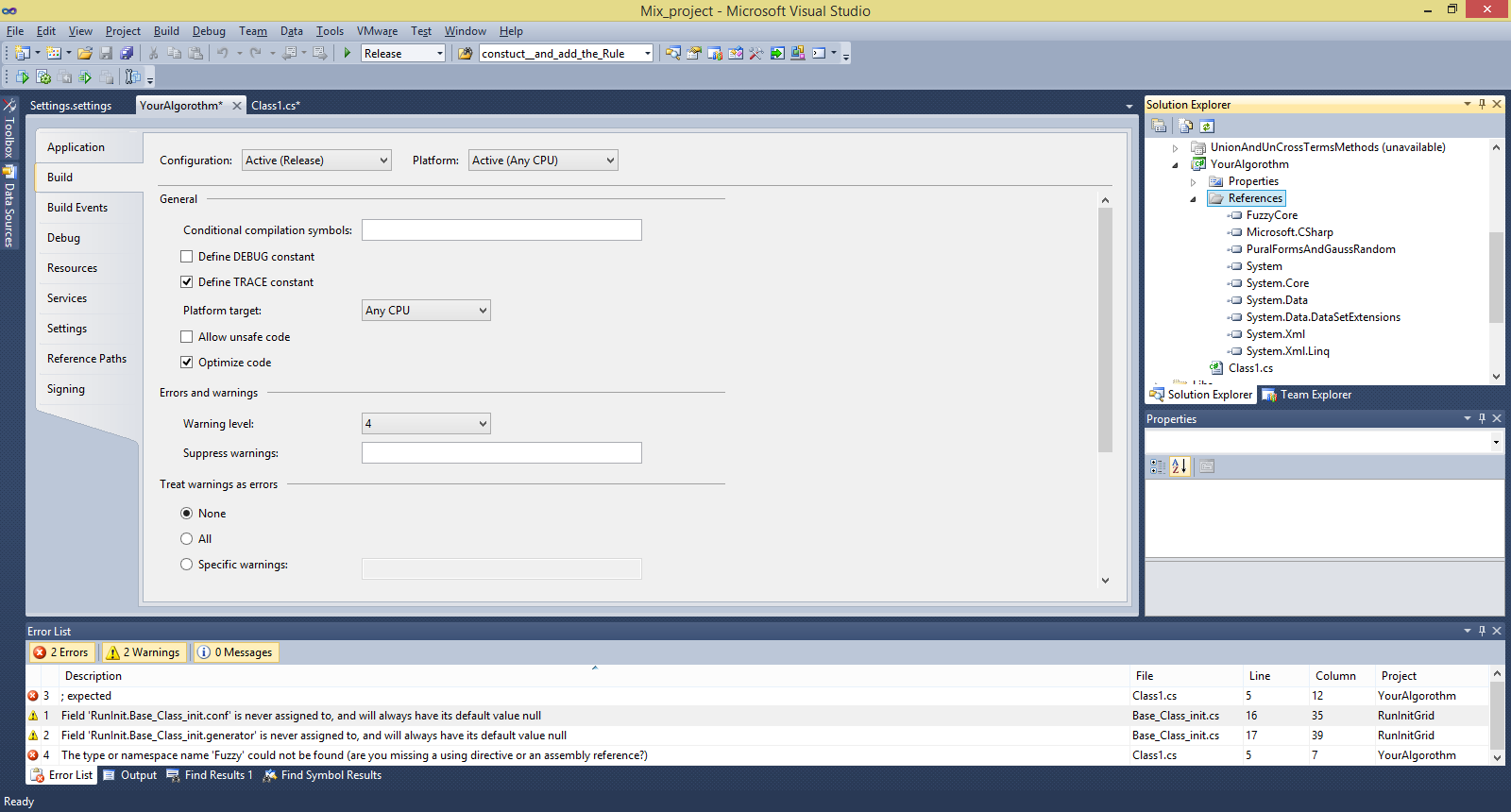
**Акт 1. С чего начать.**

1. Загрузить решение Mix\_Core.sln
2. Добавить новый проект (Библиотека классов) в ветку LearnMethods решения
3. 
4. Добавить в проект файл параметров Settings.settings



1. Перенести созданный Settings.settings в папку Properties вашего проекта
2. В свойствах проекта(Properties) на вкладке Построение (Build ) указать папку выхода в Mix\_core\GUI\FormsBoth\bin\Release\Methods\Tune
3. В сборки(Reference) добавить FuzzyCore и PuralFormsAndGaussRandom.



1. В создаваемых вами классах для работы с аппроксиматором надо в секцию using добавить

using Fuzzy\_system. Approx\_Singletone;

using Fuzzy\_system. Approx\_Singletone.learn\_algorithm;

using Fuzzy\_system.Fuzzy\_Abstract.conf;

А ваш класс должен реализовывать интерфейс Abstract\_learn\_algorithm.

Для этого должны быть методы

public a\_Fuzzy\_System TuneUpFuzzySystem(a\_Fuzzy\_System Approx, ILearnAlgorithmConf conf) // Здесь ведется оптимизация вашим алгоритмом

{

a\_Fuzzy\_System result = Approx;

//Your's code

return result;

}

public string ToString(bool with\_param = false)// без параметров возвращает имя алгоритма, с параметров true возвращает имя алгоритма и значения его параметров

{

if (with\_param)

{

string result = "Your's Algorithm Name{";

// result+= param1+Environment.NewLine;

// result+= param1+Environment.NewLine;

// result+= param1+Environment.NewLine;

result += "}";

return result;

}

return "Your's Algorithm Name";

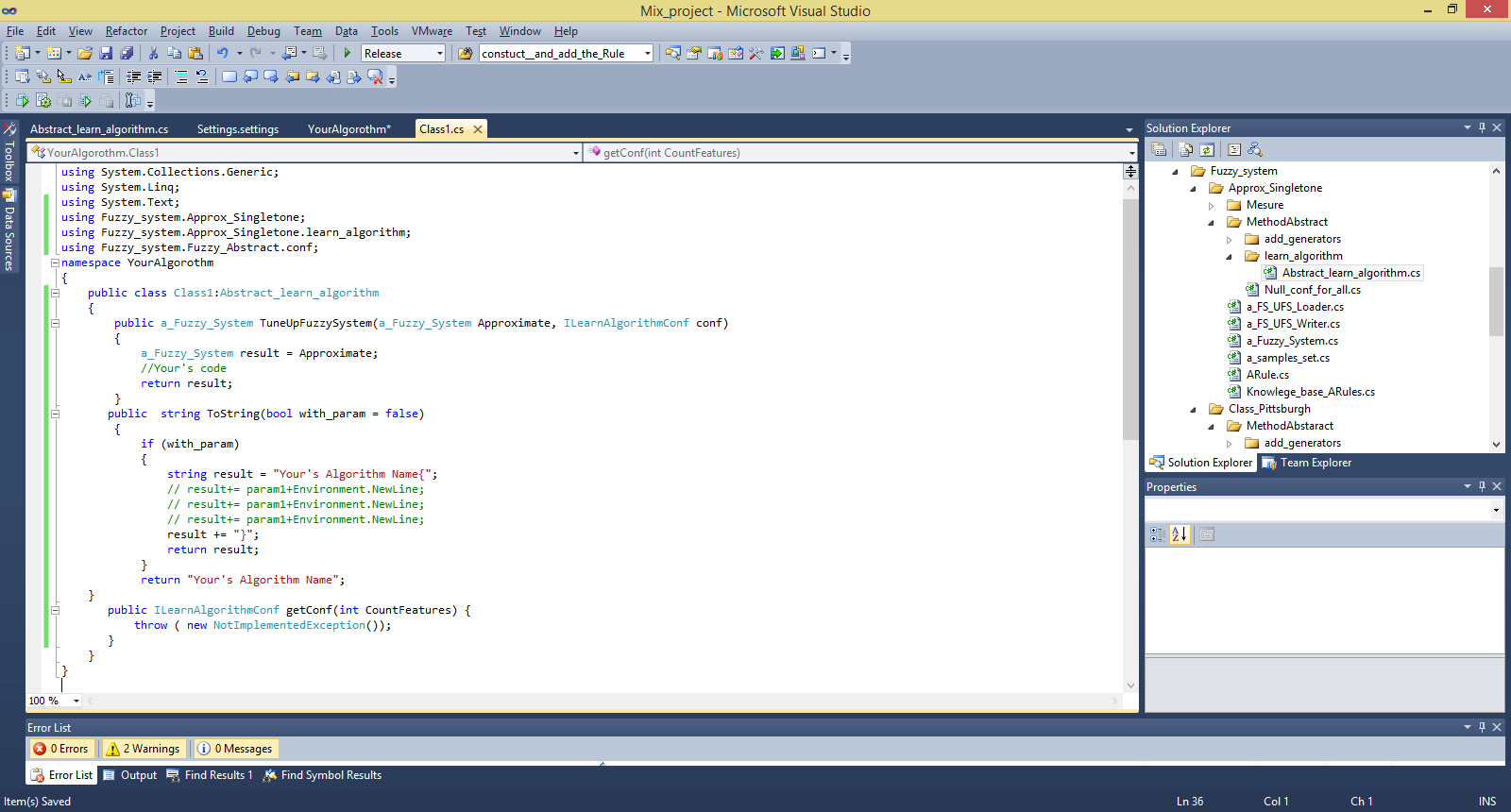
}

public ILearnAlgorithmConf getConf(int CountFeatures) // Создание класса конфигуратора для вашего метода

{

throw ( new NotImplementedException());

}



1. Для классификатора d создаваемых вами классах надо в секцию using добавить

using Fuzzy\_system.Class\_Pittsburgh;

using Fuzzy\_system.Class\_Pittsburgh.learn\_algorithm;

using Fuzzy\_system.Fuzzy\_Abstract.conf;

А ваш класс должен быть реализовывать интерфейс Abstract\_learn\_algorithm.

Для этого должны быть методы

public c\_Fuzzy\_System TuneUpFuzzySystem(c\_Fuzzy\_System Classifier, ILearnAlgorithmConf conf) // Здесь ведется оптимизация вашим алгоритмом

{

c\_Fuzzy\_System result = Classifier;

//Your's code

return result;

}

public string ToString(bool with\_param = false)// без параметров возвращает имя алгоритма, с параметров true возвращает имя алгоритма и значения его параметров

{

if (with\_param)

{

string result = "Your's Algorithm Name{";

// result+= param1+Environment.NewLine;

// result+= param1+Environment.NewLine;

// result+= param1+Environment.NewLine;

result += "}";

return result;

}

return "Your's Algorithm Name";

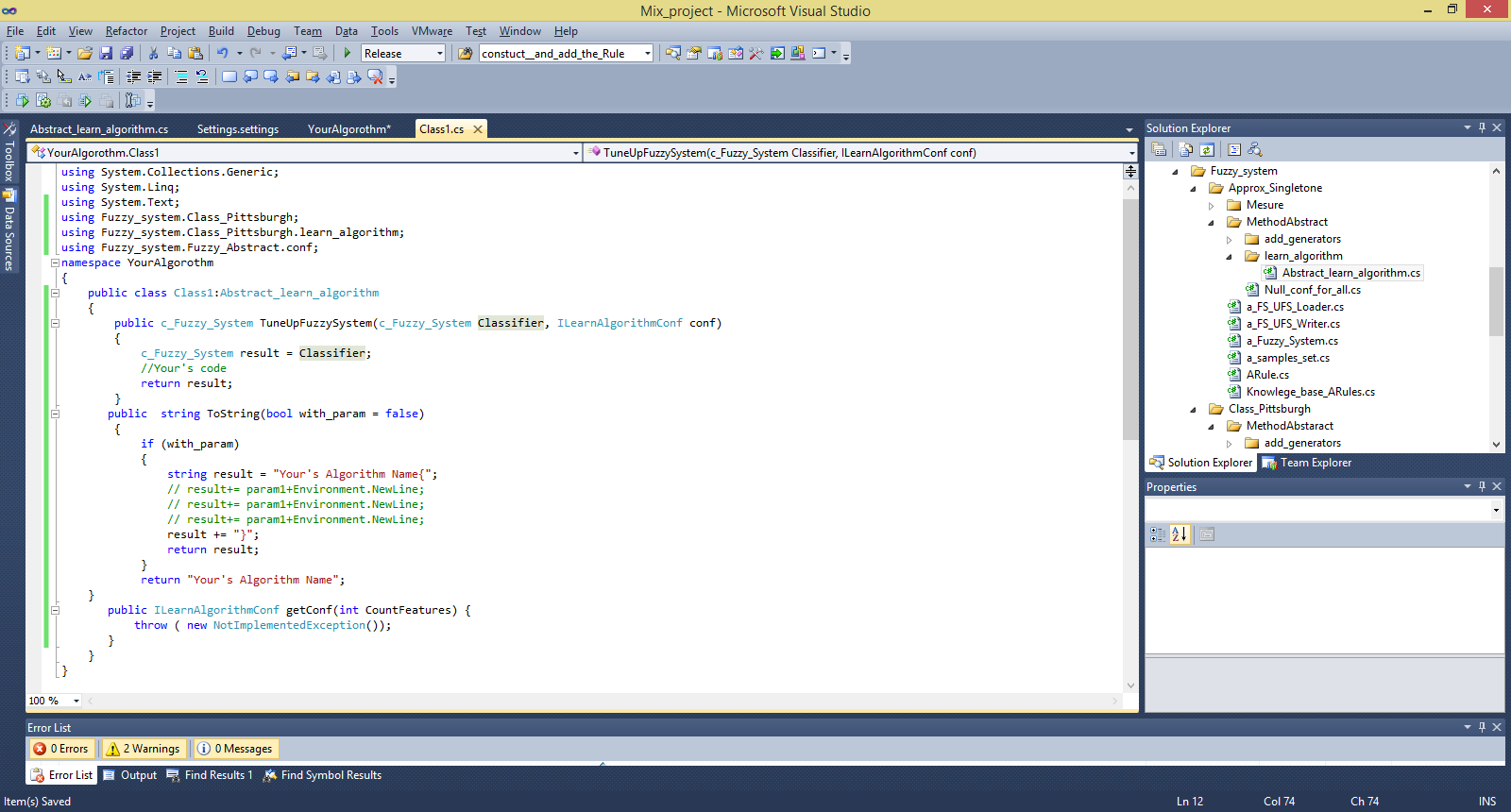
}

public ILearnAlgorithmConf getConf(int CountFeatures) // Создание класса конфигуратора для вашего метода

{

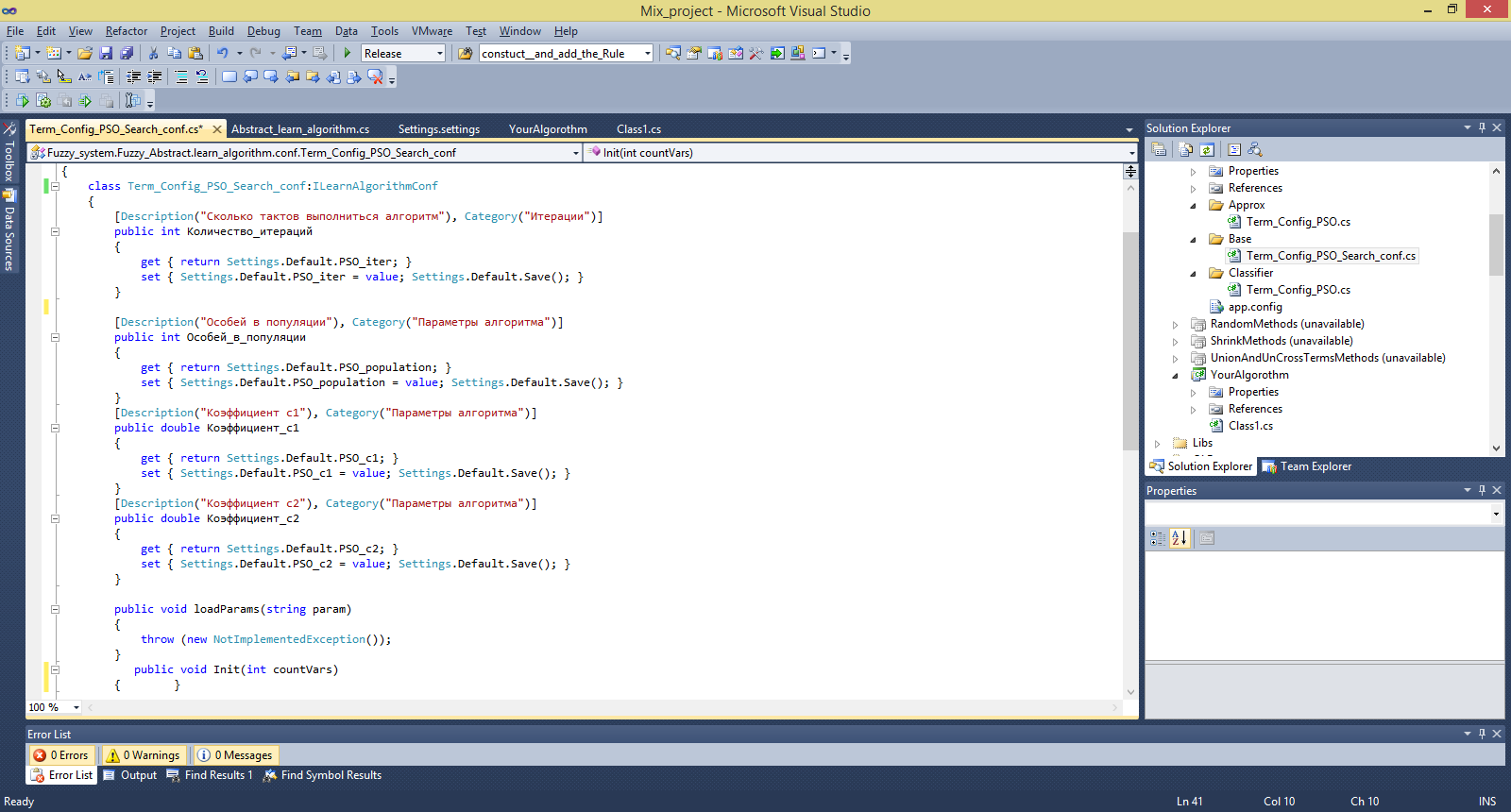
throw ( new NotImplementedException());

}



Для передачи параметров между алгоритмом и приложением необходимо создать класс конфигуратор следующего вида реализующий интерфейс ILearnAlgorithmConf

В классе должны быть объявлены публично свойства для них указаны get и set методы с сохранением в Properties.Settings.Default



Для примера подробного использования смотрите проекты EsMethods (реализован в объектном стиле) и PSOMethods ( реализован в процедурном стиле)

**Акт 2. Что из себя представляет нечеткая система, доступные поля и методы.**

В ваш метод оптимизации TuneUpFuzzySystem будет передан один из видов нечеткой системы:

* a\_Fuzzy\_System Approx- Аппроксиматор типа синглтон
* c\_Fuzzy\_System Classifier – Классификатор.

Общая часть для этих систем на примере аппроксиматора

**Поле**:

* Approx.Rulles\_Database\_Set[0] –в списке Rulles\_Database\_Set хранятся базы правил нечетких систем. В момент вызова вашего метода параметрической оптимизации можно быть уверенным только в одном в Rulles\_Database\_Set[0] будет лежать лучшая(с минимальной ошибкой) нечеткая база правил. Могул лежать еще базы правил, а может и нет.

Соглашение: После вашего алгоритма там должна лежать база правил с минимальной ошибкой созданная вашим алгоритмом или то что было до вас если полученные вами решения хуже.

* Approx.Learn\_Samples\_set – обучающая таблица наблюдений, может быть использована для определения допустимых границ изменений по каждому входному параметру;
* Approx.Count\_Rulles\_Databases – Количество нечетких баз правил в Approx.Rulles\_Database\_Set, тоже что и Approx.Rulles\_Database\_Set.Count;
* Approx.Count\_Samples – Количество строк наблюдений в таблице наблюдений, тоже что и Approx. Learn\_Samples\_set.Count;
* Approx.Count\_Vars – Количество входных параметров в таблице наблюдений, тоже что и Approx.Learn\_Samples\_set.Count\_Vars

**Методы**:

* Approx.unlaid\_protection\_fix(i) – пытается исправить *i*-тую (Approx.Rulles\_Database\_Set[i]) базу от неопределенности, проверяет пересечение термов, покрытие всей области определения, в случае неопределенности растягивает термы близкие к области неопределенности для ее покрытия. Пользоваться перед вычислением ошибки, возможно, но не рекомендуется т.к. высокие вычислительный затраты на проверку. Метод не гарантирует правильность при усечении количества правил в базе правил (Актуально для структурной оптимизации);

**Методы для аппроксиматора**:

* Approx.approx\_Learn\_Samples(i) - Возвращает ошибку(RMSE) аппроксимации для базы правил в Approx.Rulles\_Database\_Set[i] обучающей таблицы наблюдения. Помните при работе этого метода никто не должен параллельно изменять Approx.Rulles\_Database\_Set[i]. Чем больше ошибка тем хуже.
* Approx.approx\_Test\_Samples (i) - Возвращает ошибку (RMSE) аппроксимации для базы правил в Approx.Rulles\_Database\_Set[i] тестовой таблице наблюдения. Помните при работе этого метода никто не должен параллельно изменять Approx.Rulles\_Database\_Set[i]. Чем больше ошибка тем хуже.
* Approx.Nearest\_Approx( List <Term> Source) - Возвращает значение (double) консеквента вычисленное методом ближайшего соседа для правила у которого антецедент состоит из термов List <Term> Source.

**Методы для классификатора:**

* Classifier.Classify\_Learn\_Samples(i) - Возвращает процент правильно классифицированных базой правил в Classifier.Rulles\_Database\_Set[i] в обучающей таблице наблюдения. Помните при работе этого метода никто не должен параллельно изменять Classifier.Rulles\_Database\_Set[i]. Чем больше процент правильной классификации, тем лучше.
* Classifier. Classify\_Test\_Samples (i) - Возвращает процент правильно классифицированных базой правил в Classifier.Rulles\_Database\_Set[i] в тестовой таблице наблюдения. Помните при работе этого метода никто не должен параллельно изменять
* Classifier.Rulles\_Database\_Set[i]. Чем больше процент правильной классификации, тем лучше.
* Classifier. Nearest\_Class( List <Term> Source) - Возвращает идентификатор класса (string) консеквент найденный методом ближайшего соседа для правила у которого антецедент состоит из термов List <Term> Source.

**Акт 3. Строение нечеткой базы правил**

База правил аппроксиматора представленная в Approx.Rulles\_Database\_Set[0] является объектом класса Knowlege\_base\_ARules и для классификатора Classifier.Rulles\_Database\_Set[0] объектом класса Knowlege\_base\_СRules

Рассмотрим их общие поля и методы.

**Поля**:

* Rulles\_Database\_Set[0].Terms\_Set – только возвращает весь список термов в базе правил без привязки к правилам. Однако, изменение объектов Terms\_Set[i] влияет на все правил в которые входит терм.
* Rulles\_Database\_Set[0].Rules\_Database - только возвращает весь список правил в базе правил. Позволяет обратиться к одному конкретному правилу, его термам и консеквенту. Изменение терма правила повлияет изменит тот же самый терм во всех правилах в которые он входит.

**Поля для аппроксиматора**:

* Rulles\_Database\_Set[0].all\_conq\_of\_rules – Свойство возвращает или устанавливает массив фиксированной длины типа double [n]. В массиве все консеквенты правил

**Поля для классификатора**:

* Rulles\_Database\_Set[0].Weigths – Свойство возвращает или устанавливает массив фиксированной длины типа double [n]. В массиве все веса правил Питтсбургского классификатора.

**Методы общие для аппроксиматора и классификатора**:

Конструктор для клонов нечетких баз правил new Knowlege\_base\_ARules(Knowlege\_base\_ARules Source) – создает и возвращает новую базу правил для аппроксиматора(Для классификатора) идентичную нечеткой базе правил Source.

Замечание используйте данный метод чтобы избежать ошибок новичка.

*Пример* :

1. Knowlege\_base\_ARules A = Rulles\_Database\_Set[0]; Как бы скопировали лучшее для последующей оптимизации;
2. A= YouAlgorithmOptimize(A); //Оптимизировали
3. Rulles\_Database\_Set.Add(A); // Добавили новую базу правил в список баз правил нечеткой системы.
4. double ErrorBefore = Approx.approx\_Learn\_Samples(0); // Как бы посмотрели ошибку до оптимизации;
5. double ErrorAfter = Approx.approx\_Learn\_Samples(1) ; Посмотрели ошибку после оптимизации.

НО! В этом примере ErrorBefore всегда будет равен ErrorAfter. Т.к. Knowlege\_base\_ARules A – всего лишь указатель на базу правил Rulles\_Database\_Set[0], а не новая база правил. В момент изменения A вы меняете и Rulles\_Database\_Set[0];

*Правильная версия выглядит вот так* :

1. Knowlege\_base\_ARules A = new Knowlege\_base\_ARules( Rulles\_Database\_Set[0]); Создали клона базы правил чтобы дальше его менять;
2. A= YouAlgorithmOptimize(A); //Оптимизировали
3. Rulles\_Database\_Set.Add(A); // Добавили новую базу правил в список баз правил нечеткой системы.
4. double ErrorBefore = Approx.approx\_Learn\_Samples(0); // Посмотрели ошибку до оптимизации;
5. double ErrorAfter = Approx.approx\_Learn\_Samples(1) ; Посмотрели ошибку после оптимизации у когда то клонированной а потом и измененной базы правил.

**Акт 4. Из чего же состоит нечеткое правило?**

* Rulles\_Database\_Set[0].Rules\_Database[0] – правило нечеткой системы объект типа ARule для аппроксиматора и CRule для классификатора.

**Поля:**

* Rules\_Database[0].Term\_of\_Rule\_Set – список ссылок на термы из Rulles\_Database\_Set[0].Terms\_Set. Изменение Rules\_Database[0].Term\_of\_Rule\_Set[0] изменит тем во всех правила где указан, не рекомендуется использовать для изменения терма. Лучше ограничиться чтением параметром терма если необходимо.

**Для аппроксиматора:**

* Rules\_Database[0].Cons\_approx\_Value – значение консеквента(double) правила аппроксиматора. Доступно для чтение и записи.

**Для классификатора:**

* Rules\_Database[0].Label\_of\_Class – значение идентификатора класса(string) правила аппроксиматора. Доступно для чтение и записи.
* Rules\_Database[0].CF – значение веса правила классификатора(double). Доступно для чтения из записи.

**Акт 5. Загадочный нечеткий терм.**

И

* Rulles\_Database\_Set[0].Rules\_Database[0].Term\_of\_Rule\_Set и Rulles\_Database\_Set[0]. Terms\_Set используют для работы понятие нечеткого терма в программе он представлен объектами класса Term.

**Поля терма:**

* Terms\_Set[0].Term\_Func\_Type – тип функции принадлежности, на текущий момент может быть треугольной гауссоидой, параболой и трапеций возможно будут и дальше расширен список.

Именно от этого поля зависит и количество параметров терма и как он будет вычислять уровень принадлежности.

*Пример проверки типа терма и последующих различных действий для каждого типа*:

switch(Terms\_Set[0].Term\_Func\_Type)

{

case Type\_Term\_Func\_Enum.Гауссоида: {); } break;

case Type\_Term\_Func\_Enum.Парабола: {} break;

case Type\_Term\_Func\_Enum.Трапеция: {} break;

case Type\_Term\_Func\_Enum.Треугольник: {} break;

}

* Terms\_Set[0].Parametrs – значение свойств параметров терма именно это поле вам и придется оптимизировать. Всегда массив double заданной длины. Узнать размерность массива параметров можно Terms\_Set[0].Parametrs.Count(); или Member\_Function.Count\_Params\_For\_Term(Terms\_Set[0].Term\_Func\_Type);

Смысл параметров кодируется согласно их расположению описан в таблице ниже:

Соответствие номеров, параметрам разных функций принадлежности

Граница – это координата точки пересечения с осью OX;

Вершина – это Координата точки, где уровень принадлежности равен 1.

Треугольник(Triangle):

|  |  |  |
| --- | --- | --- |
| Параметр | Номер параметра | Описание |
| A | 0 | левая граница |
| B | 1 | вершина треугольника |
| C | 2 | правая граница треугольника |

Трапеция(Trapezoid):

|  |  |  |
| --- | --- | --- |
| Параметр | Номер параметра | Описание |
| a | 0 | левая граница |
| b | 1 | левая вершина трапеции |
| c | 2 | правая вершина трапеции |
| d | 3 | правая граница |

Гаусс(Gauss):

|  |  |  |
| --- | --- | --- |
| Параметр | Номер параметра | Описание |
| a | 0 | центр функции Гаусса |
| g | 1 | коэффициент разброса относительно центра |

Парабола(Parabolic):

|  |  |  |
| --- | --- | --- |
| Параметр | Номер параметра | Описание |
| a | 0 | Левая граница |
| b | 1 | Правая граница |

Продолжаем о полях терма

* Terms\_Set[0].Number\_of\_Input\_Var – указывает(int) для какой по номеру входного параметра таблицы наблюдения используется данный терм. Только для чтения.
* Terms\_Set[0].Max – указывает(double) на значение правой границы терма. Рекомендуется использовать только для чтения. Для гауссоиды фиктивная величина равная значения функции гаусса + 3 значения коэффицента разброса.
* Terms\_Set[0].Min – указывает(double) на значение правой границы терма. Рекомендуется использовать только для чтения. Для гауссоиды фиктивная величина равная значения функции гаусса - 3 значения коэффицента разброса.
* Terms\_Set[0].Pick – указывает(double) на значение вершины терма. Рекомендуется использовать только для чтения.

**Послесловие.** Это всего лишь черновая версия, описания сделанная на скорую руку, здесь возможны орфографические, лексические, грамматические и другие ошибки.