ЗАДАНИЕ

Разработать систему классов для решения задачи оптимизации на базе следующих интерфейсов.

1. **IParametricFunction** - параметрическая функция, Bind фиксирует параметры и возвращает следующие реализации:
   - *Линейная функция в n-мерном пространстве* (число параметров n+1, реализует `IDifferentiableFunction`;
   - *Полином n-й степени в одномерном пространстве* (число параметров n+1, не реализует `IDifferentiableFunction`);
   - *Кусочно-линейная функция* (реализует `IDifferentiableFunction`);
   - *Сплайн* (не линейный).
2. **IFunctional** - минимизируемый функционал, должны быть следующие реализации:
   - L_1 *норма разности* с требуемыми значениями в наборе точек (реализует `IDifferentiableFunctional`, не реализует `ILeastSquaresFunctional`);
   - L_2 *норма разности* с требуемыми значениями в наборе точек (реализует `IDifferentiableFunctional`, реализует `ILeastSquaresFunctional`);
   - L_{inf} *норма разности* с требуемыми значениями в наборе точек (не реализует `IDifferentiableFunctional`, не реализует `ILeastSquaresFunctional`);
   - *Интеграл по некоторой области* (численно).
3. **IOptimizator** - метод минимизации, должны быть следующие реализации:

- (универсальный) *Метод Монте-карло* (лучше *алгоритм имитации отжига*);
- (требующий `IDifferentiableFunctional`) *Метод градиентного спуска* (лучше *метод сопряжённых градиентов*);
- (требующий `ILeastSquaresFunctional`) *Алгоритм Гаусса-Ньютона*.

Обязательные требования:

- Интерфейсы из задания менять нельзя;
- Классы должны взаимодействовать только через эти интерфейсы:

```
namespace interfaceexample
{
    interface IVector : IList<double> { }

    interface IParametricFunction
    {
        IFunction Bind(IVector parameters);
    }

    interface IFunction
    {
        double Value(IVector point);
    }

    interface IDifferentiableFunction : IFunction
    {
        // По параметрам исходной IParametricFunction
        IVector Gradient(IVector point);
    }
    interface IFunctional
    {
        double Value(IFunction function);
    }
    interface IDifferentiableFunctional : IFunctional
    {
        IVector Gradient(IFunction function);
    }
```

```
interface IMatrix : IList<IList<double>> {}
interface ILeastSquaresFunctional : IFunctional
{
    IVector Residual(IFunction function);
    IMatrix Jacobian(IFunction function);
}
interface IOptimizator
{
    IVector Minimize(IFunctional objective, IParametricFunction function, IVector
    ↪  initialParameters, IVector minimumParameters = default, IVector
    ↪  maximumParameters = default);
}
public class Vector : List<double>, IVector {}
class LineFunction : IParametricFunction
{
    class InternalLineFunction : IFunction
    {
        public double a, b;
        public double Value(IVector point) => a * point[0] + b;
    }
    public IFunction Bind(IVector parameters) => new InternalLineFunction() { a =
    ↪  parameters[0], b = parameters[1] };
}

class MyFunctional : IFunctional
{
    public List<(double x, double y)> points;
    public double Value(IFunction function)
    {
        double sum = 0;
        foreach (var point in points)
        {
            var param = new Vector();
            param.Add(point.x);
            var s = function.Value(param) - point.y;
            sum += s * s;
        }
        return sum;
    }
}
```

```
class MinimizerMonteCarlo : IOptimizator
{
    public int MaxIter = 100000;
    public IVector Minimize(IFunctional objective, IParametricFunction function,
    ↪  IVector initialParameters, IVector minimumParameters = null, IVector
    ↪  maximumParameters = null)
    {
        var param = new Vector();
        var minparam = new Vector();
        foreach (var p in initialParameters) param.Add(p);
        foreach (var p in initialParameters) minparam.Add(p);
        var fun = function.Bind(param);
        var currentmin = objective.Value(fun);
        var rand = new Random(0);
        for (int i = 0; i < MaxIter; i++)
        {
            for (int j = 0; j < param.Count; j++) param[j] = rand.NextDouble();
            var f = objective.Value(function.Bind(param));
            if (f < currentmin)
            {
                currentmin = f;
                for (int j = 0; j < param.Count; j++) minparam[j] = param[j];
            }
        }
        return minparam;
    }
}

class Program
{
    static void Main(string[] args)
    {
        var optimizer = new MinimizerMonteCarlo();
        var initial = new Vector();
        initial.Add(1);
        initial.Add(1);
        int n = int.Parse(Console.ReadLine());
        List<(double x, double y)> points = new();
        for (int i = 0; i < n; i++)
        {
```

```
                var str = Console.ReadLine().Split();
                points.Add((double.Parse(str[0]), double.Parse(str[1])));
            }
            var functinal = new MyFunctional() { points = points };
            var fun = new LineFunction();

            var res = optimizer.Minimize(functinal, fun, initial);
            Console.WriteLine($"a={res[0]},b={res[1]}");
        }
    }
}
```

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Стоит ли выдавать базу про архитектуру?

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

Надо ли?

2.1. ФУНКЦИОНАЛЫ

2.2. ФУНКЦИИ

При реализации функций в основе лежали следующие идеи их представления.

- *Линейная функция в n-мерном пространстве* будет искаться в виде $f(\overline{x}) = c_0 + c_1 \cdot x_1 + c_2 \cdot x_2 + ... + c_n \cdot x_n$. Вектор входных параметров для данной функции выглядит следующим образом: $[c_0, c_1, ..., c_n]$.
- *Полином n-й степени в одномерном пространстве* будет искаться в виде $P(x) = c_0 + c_1 \cdot x + c_2 \cdot x^2 + ... + c_n \cdot x^n$. Вектор входных параметров для данной функции выглядит следующим образом: $[c_0, c_1, ..., c_n]$.
- *Кусочно-линейную функции* представим в виде системы уравнений $a \cdot x + b \cdot y = c, \forall x \in [x_i, x_{i+1}]$. Тогда вектор входных параметров будет иметь вид: $[x0, x1, ...x_{n-1}, a_0, a_1, ...a_n, b_0, b_1, ...b_n, c_0, c_1, ...c_n]$, где $[x_0, x_1, ...x_{n-1}]$ - координаты точек разлома, $[a_0, a_1, ...a_n]$ - коэффициенты при x, $[b_0, b_1, ...b_n]$ - коэффициенты при y, $[c_0, c_1, ...c_n]$ - свободные коэффициенты.
- *Сглаживающий сплайн* вида $S(x) = \sum_{i=0}^{2n} q_i \cdot \psi_i(x)$ на эрмитовых базисных функциях. Вектор входных параметров определяется видом: $[q_0, ..., q_{2n}, x_0, ..., x_n]$.

2.3. МЕТОДЫ ОПТИМИЗАЦИИ

В качестве методов оптимизации были реализованы следующие алгоритмы:

- универсальный – *Алгоритм имитации отжига*;
- требующий `IDifferentiableFunctional` – *Метод сопряжённых градиентов*;
- требующий `ILeastSquaresFunctional` – *Алгоритм Левенберга — Марквардта.*

3. ТЕСТИРОВАНИЕ

Проведем несколько тестов на правильность решения задачи оптимизации.

3.1. ТЕСТИРОВАНИЕ РАЗЛИЧНЫХ ФУНКЦИОНАЛОВ

3.2. ТЕСТИРОВАНИЕ РАЗЛИЧНЫХ ФУНКЦИЙ

3.3. ТЕСТИРОВАНИЕ РАЗЛИЧНЫХ МЕТОДОВ ОПТИМИЗАЦИИ