Лабораторная работа 2 Методы многомерной безусловной оптимизации	ФИО студента	Титов А.К.
	Группа	ИВТ - 261
	Методы	2.1.2 (Метод Хука - Дживса) Метод случайного поиска
	Задачи	2.1
	Дата отчета	
	Оценка	
	Подпись преподавателя	

Задание

Реализовать программно методы многомерной безусловной оптимизации

- Хука - Дживса
- Случайного поиска

С их помощью решить тестовую задачу 2.1

Описание решения

Были написаны классы **VectorN** (N – мерный вектор), **PointN** (N – мерная точка).
С их помощью были реализованы заявленные в задании алгоритмы.
Была изменена структура этих алгоритмов (сведены к основным элементам, используемым в структурном программировании).

Задача 2.1

2.1. Тестовая задача

Минимизировать функцию Химмельблау №1 $f(x) = 4(x_1 - 5)^2 + (x_2 - 6)^2$.
(3D графики функции с различными углами вращения и наклона осей переменных приведены на рис. 6).
Начальные данные: $x^0 = (5,6)^T$ и $\varepsilon = 0.01$, шаги (для методов прямого поиска) $h_1 = h_2 = 1$.
Ожидаемый результат: $x^* \approx (1,1)^T$, $f^* \approx 0$ (приближенное решение).

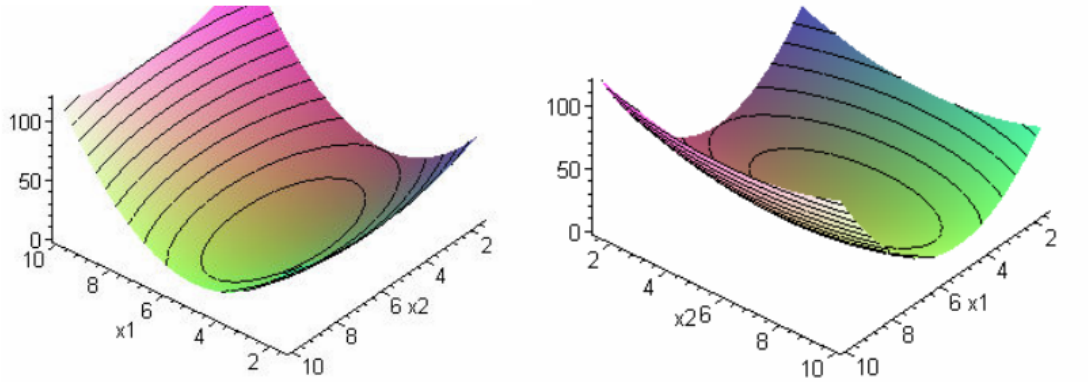
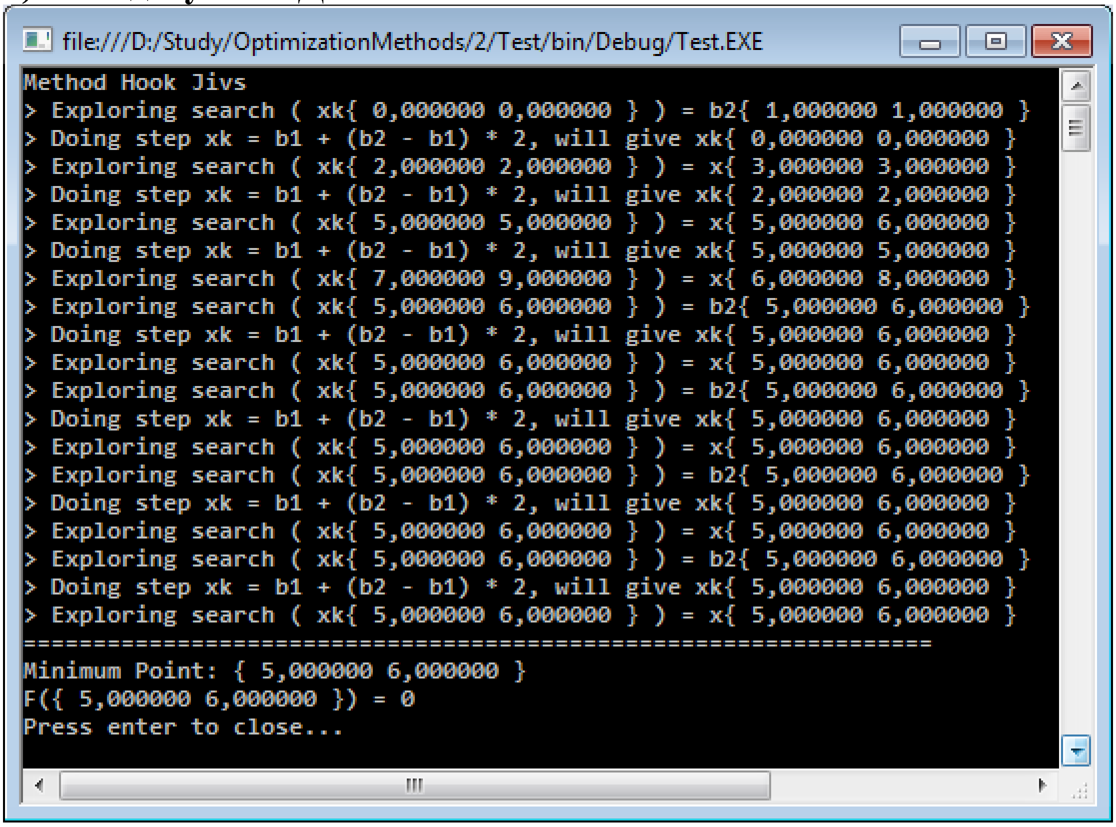


Рис. 6

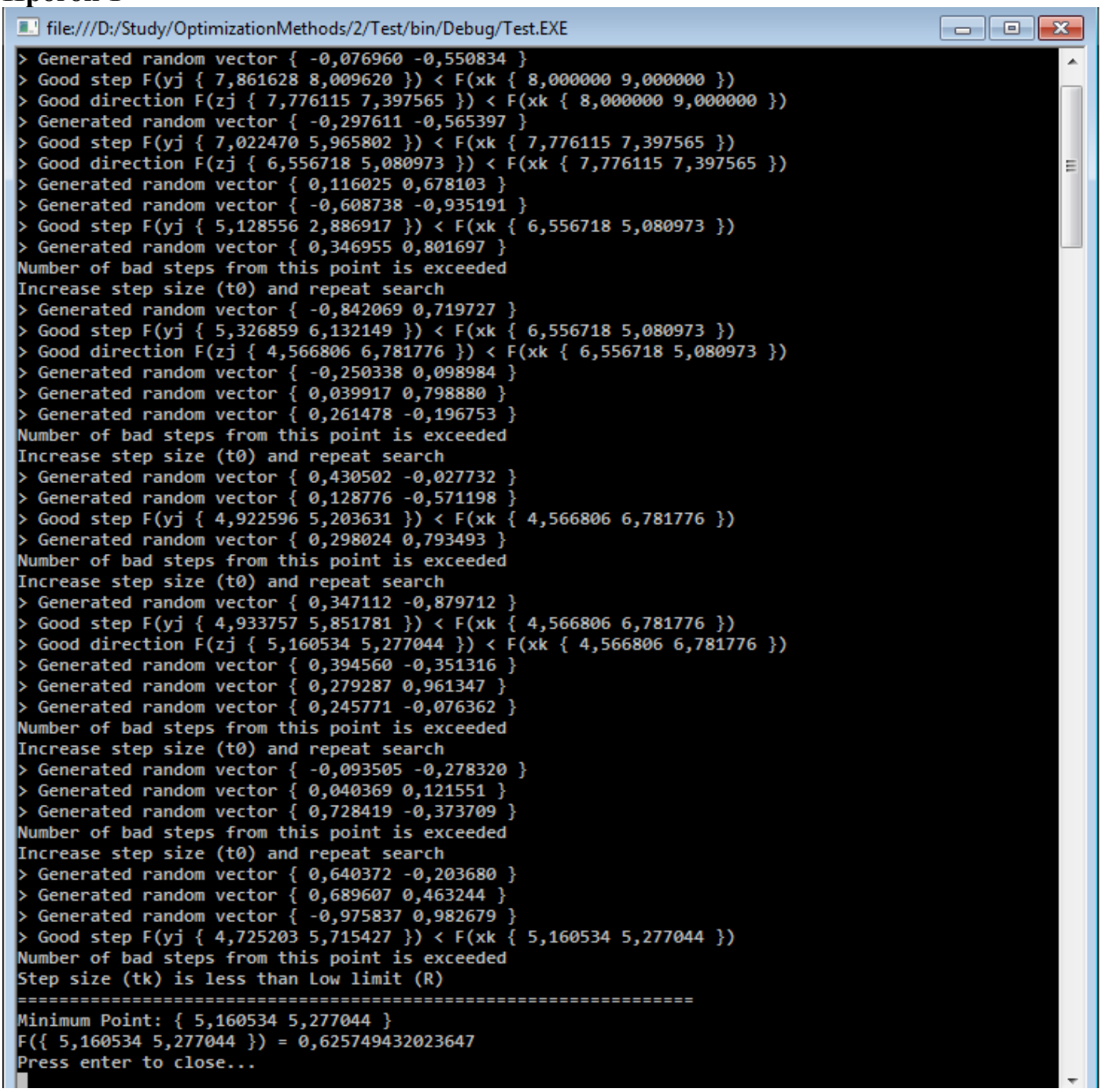
Результаты работы программ на тестовой задаче 2.1

1) Метод Хука — Дживса

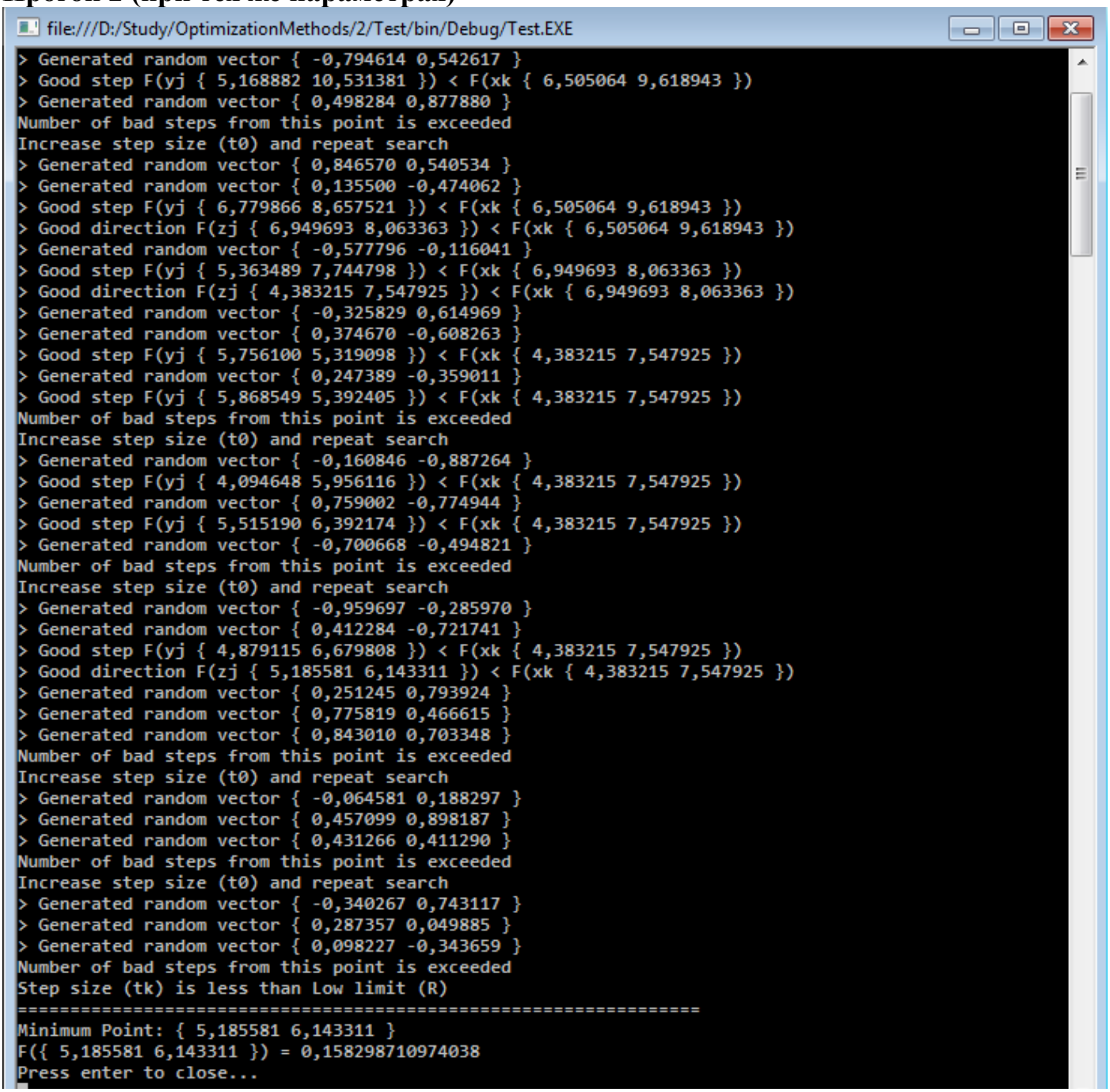


```
file:///D:/Study/OptimizationMethods/2/Test/bin/Debug/Test.EXE
Method Hook Jivs
> Exploring search ( xk{ 0,000000 0,000000 } ) = b2{ 1,000000 1,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 0,000000 0,000000 }
> Exploring search ( xk{ 2,000000 2,000000 } ) = x{ 3,000000 3,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 2,000000 2,000000 }
> Exploring search ( xk{ 5,000000 5,000000 } ) = x{ 5,000000 6,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 5,000000 5,000000 }
> Exploring search ( xk{ 7,000000 9,000000 } ) = x{ 6,000000 8,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = b2{ 5,000000 6,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = x{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = b2{ 5,000000 6,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = x{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = b2{ 5,000000 6,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = x{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = b2{ 5,000000 6,000000 }
> Doing step xk = b1 + (b2 - b1) * 2, will give xk{ 5,000000 6,000000 }
> Exploring search ( xk{ 5,000000 6,000000 } ) = x{ 5,000000 6,000000 }
================================================================
Minimum Point: { 5,000000 6,000000 }
F({ 5,000000 6,000000 }) = 0
Press enter to close...
```

Метод случайного поиска
Прогон 1



Прогон 2 (при тех же параметрах)



Код программы
VectorN.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MethodHookJivs
{
  public class VectorN
  {
    public VectorN(int size)
    {
      Components = new List<double>(size);
      for (int i = 0; i < size; i++)
        Components.Add(0.0);
    }

    public VectorN(List<double> components)
    {
      Components = components;
    }

    public VectorN(params double[] components)
    {
      int dimensionsCount = components.Length;
      Components = new List<double>(components.Length);
      for (int i = 0; i < dimensionsCount; i++)
        Components.Add(components[i]);
    }

    public List<double> Components { get; set; }

    public double Length
    {
      get
      {
        double summOfSquares = 0.0;
        foreach (var c in Components)
          summOfSquares += Math.Pow(c, 2);

        return Math.Sqrt(summOfSquares);
      }
    }

    public static VectorN operator *(VectorN multiplicand, double multiplier)
    {
      int dimensionsCount = multiplicand.Components.Count;
      VectorN composition = new VectorN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        composition.Components[i] = multiplicand.Components[i] * multiplier;

      return composition;
    }

    public static VectorN operator /(VectorN dividend, double divider)
    {
      int dimensionsCount = dividend.Components.Count;
      VectorN quotient = new VectorN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        quotient.Components[i] = dividend.Components[i] / divider;

      return quotient;
    }

    public string ToString()
    {
      string point = "{ ";
      foreach (var component in Components)
        point += component.ToString("N6") + ' ';

      point += "}";
      return point;
    }
  }
}
```

PointN.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MethodHookJivs
{
  public class PointN
  {
    public PointN(int size)
    {
      Coordinates = new List<double>(size);
      for (int i = 0; i < size; i++)
        Coordinates.Add(0.0);
    }

    public PointN(List<double> coordinates)
    {
      Coordinates = coordinates;
    }

    public PointN(params double[] coordinates)
    {
      int dimensionsCount = coordinates.Length;
      Coordinates = new List<double>(coordinates.Length);
      for (int i = 0; i < dimensionsCount; i++)
        Coordinates.Add(coordinates[i]);
    }

    public PointN(PointN other)
    {
      int dimensionsCount = other.Coordinates.Count;
      Coordinates = new List<double>(dimensionsCount);
      for (int i = 0; i < dimensionsCount; i++)
        Coordinates.Add(other.Coordinates[i]);
    }

    public List<double> Coordinates { get; set; }

    public static PointN operator +(PointN a, PointN b)
    {
      int dimensionsCount = a.Coordinates.Count;
      PointN c = new PointN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        c.Coordinates[i] = a.Coordinates[i] + b.Coordinates[i];

      return c;
    }

    public static PointN operator -(PointN a, PointN b)
    {
      int dimensionsCount = a.Coordinates.Count;
      PointN c = new PointN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        c.Coordinates[i] = a.Coordinates[i] - b.Coordinates[i];

      return c;
    }

    public static PointN operator *(PointN multiplicand, double multiplier)
    {
      int dimensionsCount = multiplicand.Coordinates.Count;
      PointN composition = new PointN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        composition.Coordinates[i] = multiplicand.Coordinates[i] * multiplier;

      return composition;
    }

    public static PointN operator +(PointN point, VectorN vector)
    {
      int dimensionsCount = point.Coordinates.Count;
      PointN resultPoint = new PointN(dimensionsCount);

      for (int i = 0; i < dimensionsCount; i++)
        resultPoint.Coordinates[i] = point.Coordinates[i] + vector.Components[i];

      return resultPoint;
    }


    public string ToString()
    {
      string point = "{ ";
      foreach (var coordinate in Coordinates)
        point += coordinate.ToString("N6") + ' ';

      point += "}";
      return point;
    }
  }
}
```

HookJivs.cs

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;

namespace MethodHookJivs
{
  public static class HookJivs
  {
    public static PointN MethodHookJivs(F function, PointN b1, VectorN h, double eps, double z = 0.1)
    {
      PointN x;
      do
      {
        do
        {
          PointN xk = b1;
          PointN b2 = HookJivsHelper.ExploringSearch(function, xk, h);
          Console.WriteLine("> Exploring search ( xk{0} ) = b2{1}", xk.ToString(), b2.ToString());

          do
          {
            Console.WriteLine("> Doing step xk = b1 + (b2 - b1) * 2, will give xk{0}", xk.ToString());
            xk = b1 + (b2 - b1) * 2;
            x = HookJivsHelper.ExploringSearch(function, xk, h);
            Console.WriteLine("> Exploring search ( xk{0} ) = x{1}", xk.ToString(), x.ToString());
            b1 = b2;
            b2 = x;
          } while (function.Value(x) < function.Value(b1));

        } while (function.Value(x) > function.Value(b1));


        if (h.Length <= eps)
          break;

        h = h * z;
      }while(true);

      return b1;
    }
  }

  class HookJivsHelper
  {
    public static PointN ExploringSearch(F function, PointN basisPoint, VectorN h)
    {
      double fb = function.Value(basisPoint);
      int dimensionsCount = basisPoint.Coordinates.Count;
      for (int i = 0; i < dimensionsCount; i++)
      {
        // hi * ei
        PointN hi_ei = new PointN(dimensionsCount);
        hi_ei.Coordinates[i] = h.Components[i];
        double f = function.Value(basisPoint + hi_ei);

        if (f < fb)
        {
          basisPoint = basisPoint + hi_ei;
          fb = f;
        }
        else
        {
          f = function.Value(basisPoint - hi_ei);
          if (f < fb)
          {
            basisPoint = basisPoint - hi_ei;
            fb = f;
          }
        }
      }

      return basisPoint;
    }
  }
}
```

RandomSearch.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using MethodHookJivs;

namespace MethodRandomSearch
{
  /// <summary>
  /// Причина ошибок может быть заключено в условии завершения работы алгоритма
  /// Оно определено лишь максимальным числом итераций
  /// </summary>
  public static class RandomSearch
  {
    // To make real safe random
    static RandomSearch()
    {
      random = new Random();
    }
    private static Random random;


    /// <summary>
    /// Метод случайного поиска для функции n переменных
    /// </summary>
    /// <param name="basicPoint">Начальная точка x0</param>
    /// <param name="a">Коэффициент расширения (1;+inf): 1.618 </param>
    /// <param name="b">Коэффициент сжатия (0;1): 0.618 </param>
    /// <param name="M">Максимальное число неудачно выполненых испытаний на данной итерации :
(3n)</param>
    /// <param name="t0">Начальная величина шага</param>
    /// <param name="R">Минимальная величина шага</param>
    /// <param name="N">Максимальное число итераций</param>
    /// <returns>Точка минимума</returns>
    public static PointN MethodRandomSearch(F function, PointN basicPoint, double a, double b,
double t0, double R, int N, int M)
    {
      double tk = t0; // Length of k-th step
      PointN xk = basicPoint; // Опорная точка
      PointN yj; // Точки, лежащие на гиперсфере радиуса tk с центром в точке xk
      PointN zj; // Точка, получающаяся после прыжка
      int dimensionsCount = basicPoint.Coordinates.Count;
      int k = 0, j = 1;

      do
      {
        // Step 2 Make random vector
        VectorN randomVector = new VectorN(dimensionsCount);
        MakeRandomVector(randomVector); // TODO What will be after this operator?
        Console.WriteLine("> Generated random vector {0}", randomVector.ToString());

        // Step 3 Find yj
        yj = xk + (randomVector / randomVector.Length) * tk;

        // Step 4
        // Good step
        bool isCurrentStepGood = function.Value(yj) < function.Value(xk);
        bool isNextStepAlsoGood = false;
        if (isCurrentStepGood)
        {
          Console.WriteLine("> Good step F(yj {0}) < F(xk {1})", yj.ToString(), xk.ToString());
          zj = xk + (yj - xk) * a;
          isNextStepAlsoGood = function.Value(zj) < function.Value(xk);
          if (isNextStepAlsoGood)
          {
            Console.WriteLine("> Good direction F(zj {0}) < F(xk {1})", zj.ToString(),
xk.ToString());
```

```
            xk = zj;
            tk *= a;
            k++;

            bool tooManyIterations = k == N;
            if (tooManyIterations)
            {
              Console.WriteLine("Limit of iterations N = {0} exceeded", N);
              return xk;
            }

            else
              j = 1;
          }
        }

        // Bad step
        if (!isCurrentStepGood || !isNextStepAlsoGood)
        {
          bool maxNumberOfFailsReached = j == M;
          if (!maxNumberOfFailsReached)
          {
            j++;
          }
          else if (maxNumberOfFailsReached)
          {
            Console.WriteLine("Number of bad steps from this point is exceeded");
            if (tk <= R)
            {
              Console.WriteLine("Step size (tk) is less than Low limit (R)");
              return xk;
            }
            else if (maxNumberOfFailsReached && tk > R)
            {
              Console.WriteLine("Increase step size (t0) and repeat search");
              tk *= b;
              j = 1;
            }
          }
        }
      } while (true);
    }

    static void MakeRandomVector(VectorN vector)
    {
      int dimensionsCount = vector.Components.Count;
      for (int i = 0; i < dimensionsCount; i++)
        vector.Components[i] = GetRandomNumber();
    }

    /// <summary>
    /// Random value to make random vector
    /// </summary>
    /// <returns>Random value from interval [-1; 1]</returns>
    static double GetRandomNumber()
    {
      return 2 * random.NextDouble() - 1;
    }
  }
}
```

Test.cs (решение тестовой задачи)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

using MethodRandomSearch;
using MethodHookJivs;

namespace Test
{
  class Program
  {
    static void Main(string[] args)
    {
      TestRandomSearch();
      //TestHookJivs();
    }

    public static void TestHookJivs()
    {
      Console.WriteLine("Method Hook Jivs");
      PointN basicPoint = new PointN(0.0, 0.0);
      VectorN h = new VectorN(1.0, 1.0);

      const double eps = 0.01;
      F minimizedFunction = new F((point) => 4 * Math.Pow(point.Coordinates[0] - 5, 2) +
Math.Pow(point.Coordinates[1] - 6, 2));

      PointN calculatedMinimum = MethodHookJivs.HookJivs.MethodHookJivs(minimizedFunction, basicPoint, h,
eps);
      PointN expectedMinimum = new PointN(new List<double> { 1.0, 1.0 });

      Console.WriteLine("================================================================");
      Console.WriteLine("Minimum Point: {0} ", calculatedMinimum.ToString());
      Console.WriteLine("F({0}) = {1}",  calculatedMinimum.ToString(),
minimizedFunction.Value(calculatedMinimum));
      Console.WriteLine("Press enter to close...");
      Console.ReadLine();
    }

    public static void TestRandomSearch()
    {
      Console.WriteLine("Method Random Search");
      PointN basicPoint = new PointN(8.0, 9.0);

      const double alpha = 1.618,
                   betta = 0.618,
                   t0 = 1,
                   R = 0.8;
      const int N = 10,
                M = 3;
      F minimizedFunction = new F((point) => 4 * Math.Pow(point.Coordinates[0] - 5, 2) +
Math.Pow(point.Coordinates[1] - 6, 2));

      PointN calculatedMinimum = MethodRandomSearch.RandomSearch.MethodRandomSearch(minimizedFunction,
basicPoint, alpha, betta, t0, R, N, M);
      PointN expectedMinimum = new PointN(new List<double> { 1.0, 1.0 });

      Console.WriteLine("================================================================");
      Console.WriteLine("Minimum Point: {0} ", calculatedMinimum.ToString());
      Console.WriteLine("F({0}) = {1}", calculatedMinimum.ToString(),
minimizedFunction.Value(calculatedMinimum));
      Console.WriteLine("Press enter to close...");
      Console.ReadLine();
    }
  }
}
```