

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

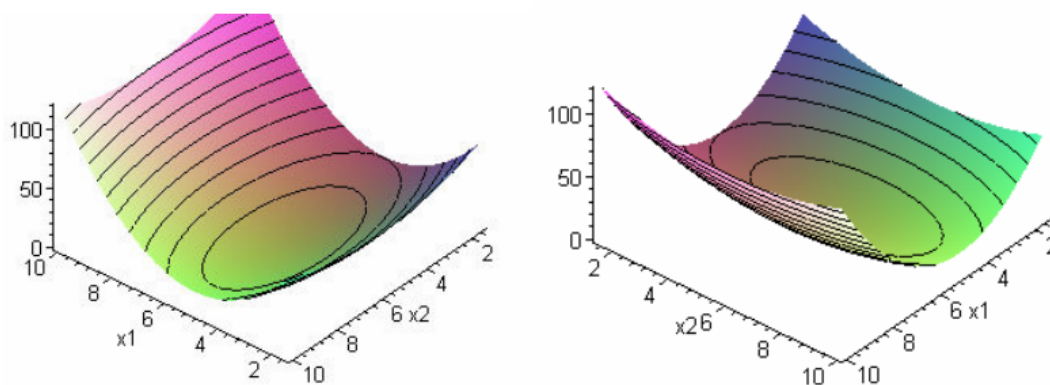
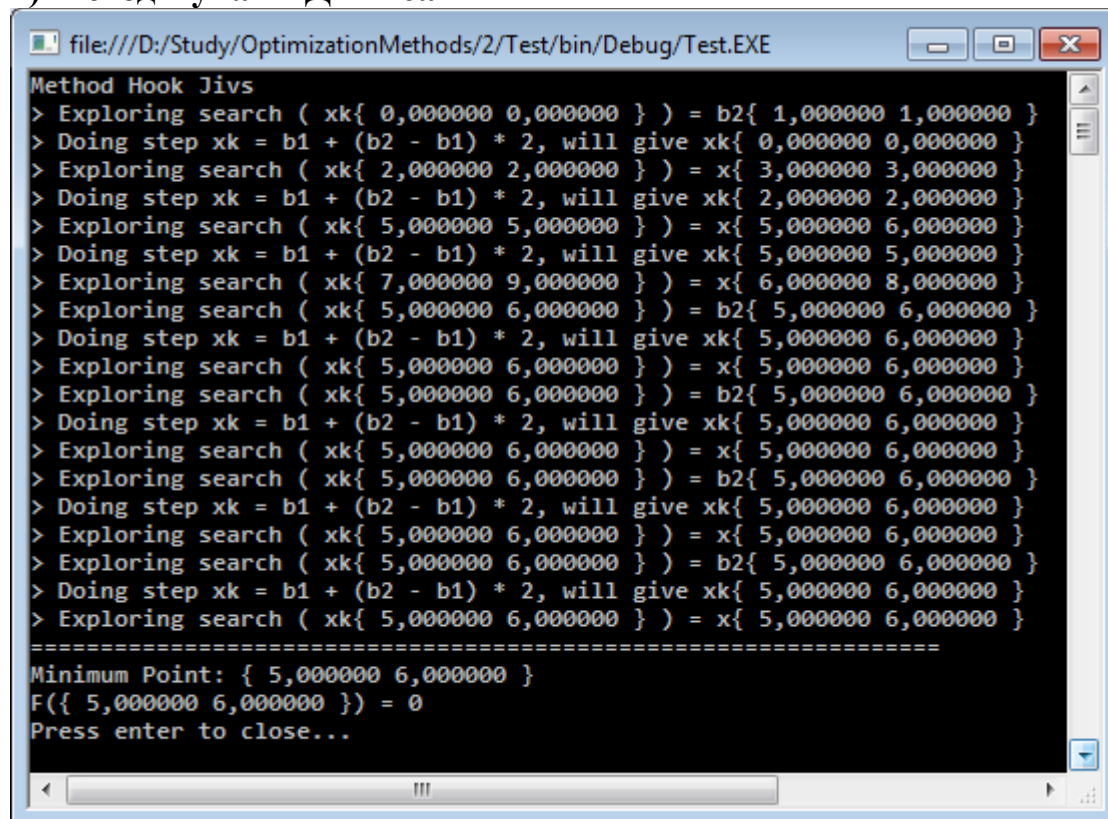


Рис. 6

## 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

```
file:///D:/Study/OptimizationMethods/2/Test/bin/Debug/Test.EXE
> Generated random vector { -0,076960 -0,550834 }
> Good step F(yj { 7,861628 8,009620 }) < F(xk { 8,000000 9,000000 })
> Good direction F(zj { 7,776115 7,397565 }) < F(xk { 8,000000 9,000000 })
> Generated random vector { -0,297611 -0,565397 }
> Good step F(yj { 7,022470 5,965802 }) < F(xk { 7,776115 7,397565 })
> Good direction F(zj { 6,556718 5,080973 }) < F(xk { 7,776115 7,397565 })
> Generated random vector { 0,116025 0,678103 }
> Generated random vector { -0,608738 -0,935191 }
> Good step F(yj { 5,128556 2,886917 }) < F(xk { 6,556718 5,080973 })
> Generated random vector { 0,346955 0,801697 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,842069 0,719727 }
> Good step F(yj { 5,326859 6,132149 }) < F(xk { 6,556718 5,080973 })
> Good direction F(zj { 4,566806 6,781776 }) < F(xk { 6,556718 5,080973 })
> Generated random vector { -0,250338 0,098984 }
> Generated random vector { 0,039917 0,798880 }
> Generated random vector { 0,261478 -0,196753 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { 0,430502 -0,027732 }
> Generated random vector { 0,128776 -0,571198 }
> Good step F(yj { 4,922596 5,203631 }) < F(xk { 4,566806 6,781776 })
> Generated random vector { 0,298024 0,793493 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { 0,347112 -0,879712 }
> Good step F(yj { 4,933757 5,851781 }) < F(xk { 4,566806 6,781776 })
> Good direction F(zj { 5,160534 5,277044 }) < F(xk { 4,566806 6,781776 })
> Generated random vector { 0,394560 -0,351316 }
> Generated random vector { 0,279287 0,961347 }
> Generated random vector { 0,245771 -0,076362 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,093505 -0,278320 }
> Generated random vector { 0,040369 0,121551 }
> Generated random vector { 0,728419 -0,373709 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { 0,640372 -0,203680 }
> Generated random vector { 0,689607 0,463244 }
> Generated random vector { -0,975837 0,982679 }
> Good step F(yj { 4,725203 5,715427 }) < F(xk { 5,160534 5,277044 })
Number of bad steps from this point is exceeded
Step size (tk) is less than Low limit (R)
=====
Minimum Point: { 5,160534 5,277044 }
F({ 5,160534 5,277044 }) = 0,625749432023647
Press enter to close...
```

## Прогон 2 (при тех же параметрах)

```
file:///D:/Study/OptimizationMethods/2/Test/bin/Debug/Test.EXE
> Generated random vector { -0,794614 0,542617 }
> Good step F(yj { 5,168882 10,531381 }) < F(xk { 6,505064 9,618943 })
> Generated random vector { 0,498284 0,877880 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { 0,846570 0,540534 }
> Generated random vector { 0,135500 -0,474062 }
> Good step F(yj { 6,779866 8,657521 }) < F(xk { 6,505064 9,618943 })
> Good direction F(zj { 6,949693 8,063363 }) < F(xk { 6,505064 9,618943 })
> Generated random vector { -0,577796 -0,116041 }
> Good step F(yj { 5,363489 7,744798 }) < F(xk { 6,949693 8,063363 })
> Good direction F(zj { 4,383215 7,547925 }) < F(xk { 6,949693 8,063363 })
> Generated random vector { -0,325829 0,614969 }
> Generated random vector { 0,374670 -0,608263 }
> Good step F(yj { 5,756100 5,319098 }) < F(xk { 4,383215 7,547925 })
> Generated random vector { 0,247389 -0,359011 }
> Good step F(yj { 5,868549 5,392405 }) < F(xk { 4,383215 7,547925 })
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,160846 -0,887264 }
> Good step F(yj { 4,094648 5,956116 }) < F(xk { 4,383215 7,547925 })
> Generated random vector { 0,759002 -0,774944 }
> Good step F(yj { 5,515190 6,392174 }) < F(xk { 4,383215 7,547925 })
> Generated random vector { -0,700668 -0,494821 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,959697 -0,285970 }
> Generated random vector { 0,412284 -0,721741 }
> Good step F(yj { 4,879115 6,679808 }) < F(xk { 4,383215 7,547925 })
> Good direction F(zj { 5,185581 6,143311 }) < F(xk { 4,383215 7,547925 })
> Generated random vector { 0,251245 0,793924 }
> Generated random vector { 0,775819 0,466615 }
> Generated random vector { 0,843010 0,703348 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,064581 0,188297 }
> Generated random vector { 0,457099 0,898187 }
> Generated random vector { 0,431266 0,411290 }
Number of bad steps from this point is exceeded
Increase step size (t0) and repeat search
> Generated random vector { -0,340267 0,743117 }
> Generated random vector { 0,287357 0,049885 }
> Generated random vector { 0,098227 -0,343659 }
Number of bad steps from this point is exceeded
Step size (tk) is less than Low limit (R)
=====
Minimum Point: { 5,185581 6,143311 }
F({ 5,185581 6,143311 }) = 0,158298710974038
Press enter to close...
```

## 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 выполненных 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,
                beta = 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, beta, 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();
        }
    }
}
```