

Лабораторная работа 3	ФИО студента	Титов А.К.
	Группа	ИВТ - 360
	Методы	2.2.1 (Оптимальный градиентный метод)
		2.2.3 (Метод переменной метрики Дэвидона-Флетчера-Пауэлла)
	Задачи	2.1.1
	Дата отчета	
	Оценка	
Градиентные методы	Подпись преподавателя	

Задание

Реализовать программно следующие градиентные методы многомерной безусловной оптимизации

- Оптимальный градиентный
- Переменной метрики Дэвидона - Флэтчера - Пауэлла

С их помощью решить тестовую задачу 2.1.1.

Описание решения

Первый алгоритм был реализован при помощи моей библиотеки NDimendionalPrimitives и алгоритмов одномерного поиска из первой лабораторной работы.

Второй алгоритм был реализован при помощи библиотеки Math.Net, так как требовал матричных вычислений.

Примечание: Полный вывод программы в протоколе не приводится из — за его объемности.

Результаты работы программ (тестовая задача 2.1)

2. Задачи численной многомерной безусловной минимизации

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: $x^0 = (0, 0)^T$ и $\varepsilon = 0.01$, шаги (для методов прямого поиска) $h_1 = h_2 = 1$.

Ожидаемый результат: $x^* \approx (5, 6)^T$, $f^* \approx 0$ (приближенное решение).

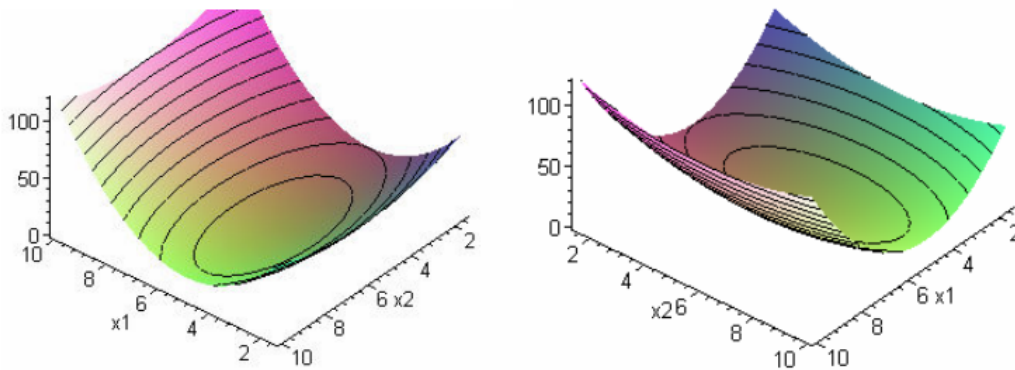


Рис. 6

Тестирующая функция

```
// Input data
FunctionNDByAlex funcND = (PointN point) =>
{
    return 4 * System.Math.Pow(point.At(0) - 5.0, 2.0) + System.Math.Pow(point.At(1) - 6.0, 2.0);
};
FunctionNDMathNet funcNDMathNet = (VectorNMathNet point) =>
{
    return 4 * System.Math.Pow(point.At(0) - 5.0, 2.0) + System.Math.Pow(point.At(1) - 6.0, 2.0);
};

PointN x0 = new PointN(1.0, 1.0);
VectorNMathNet x0MathNet = Helpers.PointNToMathNet(x0);
const double eps = 0.01;
```

```
// Testing
TestOptimalGradientMethod(funcND, x0, eps);
TestVariableMetricMethod(funcNDMathNet, x0MathNet, eps);
```

Результаты

	Метод оптимального градиента	Метод переменной метрики
Характеристика		
Время выполнения на Core i3	10 ms	365 ms
Относительная ошибка по $x[0]$, %	0,0028	0,0113
Относительная ошибка по $x[1]$, %	0,0235	0,0385

Результаты работы программ (задача 2.1.1)

2.2. Учебные задачи

2.1.1. Минимизировать функцию Химмельблау № 2 (рис. 7)

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2.$$

$x^* \approx (3.58, -1.85)^T$ – одна из точек локального минимума.

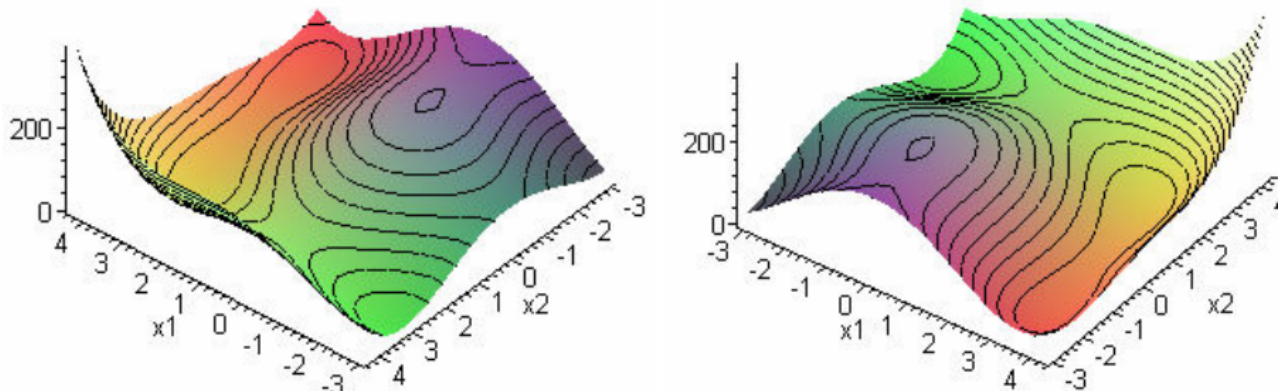


Рис. 7

Тестирующая функция

```
// Input data
FunctionNDByAlex funcND = (PointN point) =>
{
    return SMath.Pow(SMath.Pow(point.At(0), 2) + point.At(1) - 11.0, 2.0)
        + SMath.Pow(point.At(0) + SMath.Pow(point.At(1), 2) - 7.0, 2.0);
};
FunctionNDMathNet funcNDMathNet = (VectorNMathNet point) =>
{
    return SMath.Pow(SMath.Pow(point.At(0), 2) + point.At(1) - 11.0, 2.0)
        + SMath.Pow(point.At(0) + SMath.Pow(point.At(1), 2) - 7.0, 2.0);
};

PointN x0 = new PointN(3.0, 3.0);
VectorNMathNet x0MathNet = Helpers.PointNToMathNet(x0);
const double eps = 0.5; // TODO при слишком малых eps происходит деление на 0 в формуле пересчета Нк

// Testing
TestOptimalGradientMethod(funcND, x0, eps);
TestVariableMetricMethod(funcNDMathNet, x0MathNet, eps);
```

Результаты

	Метод оптимального градиента	Метод переменной метрики
Характеристика		
Время выполнения на Core i3	18 ms	287 ms
Относительная ошибка по $x[0]$, %	< 0.01	< 0.01
Относительная ошибка по $x[1]$, %	< 0.01	< 0.01

Результат работы программ на задачах:

2.1

```
file:///D:/Study/OptimizationMethods/3/Код/GradientMethods/bin/Debug/GradientMethods.EXE
Grad(F({ 1,000000 1,000000 } )) = { -32,000000 -10,000000 }
ak = 0,133911350516362
=====Next Iteration=====
Grad(F({ 5,285163 2,339114 } )) = { 2,281306 -7,321773 }
ak = 0,395133587608183
=====Next Iteration=====
Grad(F({ 4,383743 5,232192 } )) = { -4,930058 -1,535616 }
ak = 0,133910604659905
=====Next Iteration=====
Grad(F({ 5,043930 5,437827 } )) = { 0,351438 -1,124346 }
ak = 0,394641038208699
=====Next Iteration=====
Grad(F({ 4,905238 5,881540 } )) = { -0,758098 -0,236920 }
ak = 0,133954126798212
=====Next Iteration=====
Grad(F({ 5,006788 5,913276 } )) = { 0,054305 -0,173447 }
ak = 0,394380609041626
=====Next Iteration=====
Grad(F({ 4,985371 5,981681 } )) = { -0,117029 -0,036639 }
ak = 0,134000599498146
=====Next Iteration=====
Grad(F({ 5,001053 5,986590 } )) = { 0,008427 -0,026820 }
ak = 0,393851268123911
=====Next Iteration=====
Grad(F({ 4,997734 5,997153 } )) = { -0,018124 -0,005694 }
ak = 0,134025936410495
=====Next Iteration=====
Grad(F({ 5,000164 5,997916 } )) = { 0,001309 -0,004168 }
-----
Optimal gradient method: x_min = { 5,000164 5,997916 }
F(x_min) = 4,44906422573833E-06
Time: 20ms
=====
Рабочая информация метода VariableMetric лежит в txt файле VariableMetricOutput
-----
Variable metric method: x_min = {5,000005 5,999804}
F(x_min) = 3,84619994563962E-08
Time: 558ms
=====
```

2.1.1

```
file:///D:/Study/OptimizationMethods/3/Код/GradientMethods/bin/Debug/GradientMethods.EXE
Grad(F({ 3,000000 3,000000 } )) = { 22,001200 62,001200 }
ak = 0,0121732729409858
=====Next Iteration=====
Grad(F({ 2,732173 2,245242 } )) = { -12,550196 4,365794 }
ak = 0,0200297185706309
=====Next Iteration=====
Grad(F({ 2,983550 2,157797 } )) = { 1,988991 5,640432 }
ak = 0,0180440570116394
=====Next Iteration=====
Grad(F({ 2,947661 2,056021 } )) = { -2,658917 0,928511 }
ak = 0,0179721239362198
=====Next Iteration=====
Grad(F({ 2,995447 2,039333 } )) = { 0,454079 1,283789 }
ak = 0,0191467917249391
=====Next Iteration=====
Grad(F({ 2,986753 2,014753 } )) = { -0,678063 0,242268 }
ak = 0,0177699898528875
=====Next Iteration=====
Grad(F({ 2,998802 2,010448 } )) = { 0,121737 0,334648 }
ak = 0,0192536891317673
=====Next Iteration=====
Grad(F({ 2,996458 2,004005 } )) = { -0,180369 0,066475 }
ak = 0,0179273965745051
=====Next Iteration=====
Grad(F({ 2,999692 2,002813 } )) = { 0,034669 0,090460 }
-----
Optimal gradient method: x_min = { 2,999692 2,002813 }
F(x_min) = 0,000120852393519993
Time: 20ms
=====
Рабочая информация метода VariableMetric лежит в txt файле VariableMetricOutput
-----
Variable metric method: x_min = {2,998100 1,998748}
F(x_min) = 0,000207713728023837
Time: 456ms
=====
```

Код программы

OptimalGradientMethod.cs

```
namespace GradientMethods.Math
{
    using Function1D = Func<double, double>;
    using FunctionND = Func<PointN, double>;

    class OptimalGradientMethod
    {
        private FunctionND funcND;
        private PointN x0;
        public List<string> Log;
        public OptimalGradientMethod(FunctionND funcND, PointN x0)
        {
            this.funcND = funcND;
            this.x0 = new PointN(x0);

            Log = new List<string>();
        }

        public PointN FindMin(double eps)
        {
            int dimensionsCount = x0.Coordinates.Count;
            int k = 0;
            double ak = 0.0;
            PointN xk = x0;
            while(true)
            {
                VectorN gradFk = Gradient.Calculate(funcND, xk);
                Log.Add(String.Format("Grad(F({0}, -23))) = {1, -25}", xk, gradFk));

                if (gradFk.Length <= eps)
                    return xk;

                Function1D func1D = (double alpha) => { return funcND(xk - gradFk * alpha); };
                ak = OneDimensionalMinimization.FindMin(func1D, ak, eps);
                Log.Add(String.Format("ak = {0}", ak));

                xk = xk - gradFk * ak;
                k++;
                Log.Add("=====Next Iteration=====");
            }
        }
    }
}
```

Gradient.cs

```
class Gradient
{
    // Центральная разностная схема
    //      f(x+h) - f(x-h)
    // f(x) = -----
    //           2h
    public static VectorN Calculate(Func<PointN, double> f, PointN point, double h = 0.01)
    {
        int dimensionsCount = point.Coordinates.Count;
        VectorN gradient = new VectorN(dimensionsCount);

        for (int i = 0; i < dimensionsCount; i++)
        {
            PointN xMinusH = new PointN(point);
            xMinusH.Coordinates[i] -= h;
            PointN xPlusH = new PointN(point);
            xPlusH.Coordinates[i] += h;
            gradient.Components[i] = (f(xPlusH) - f(xMinusH)) / (2 * h);
        }

        return gradient;
    }
}
```

VariableMetricMethod.cs

```
namespace GradientMethods.Math
{
    using PointNMathNet = Vector<double>;
    using VectorNMathNet = Vector<double>;
    using FunctionND = Func<Vector<double>, double>;
    using Function1D = Func<double, double>;

    class VariableMetricMethod
    {
        private FunctionND funcND;
        private PointNMathNet x0;
        int n; // Размерность задачи
        Matrix<double> H0; // Единичная матрица (nхn)

        public List<string> Log;
        public VariableMetricMethod(FunctionND funcND, PointNMathNet x0)
        {
            this.funcND = funcND;
            this.x0 = x0.Clone();
            n = x0.Count;
            H0 = Matrix<double>.Build.DenseDiagonal(n, 1.0);

            Log = new List<string>();
        }

        // Это хорошая идея - выносить параметры вроде eps сюда
        // Можно добавить список строк List<String>
        // И выводить в него информацию по алгоритму
        // Затем, в output мы берем его и выводим на экран (по желанию)
        public PointNMathNet FindMin(double eps)
        {
            int k = 0;
            PointNMathNet xk = x0; // x(k)
            PointNMathNet xk_1 = x0; // x(k-1)
            Matrix<double> Hk = H0.Clone(); // Квазиньютоновская матрицы
            double ak = 0.0;

            // Вспомогательные вектора
            VectorNMathNet sk = VectorNMathNet.Build.Dense(n);
            VectorNMathNet yk = VectorNMathNet.Build.Dense(n);
            while (true)
            {
                VectorNMathNet gradient = Gradient.Calculate(funcND, xk);
                Log.Add(String.Format("Grad(F({n{0}, -23})) = {n{1}, -25}", xk, gradient));
                if (gradient.L2Norm() <= eps) // градиент слишком мал, максимум очень близко
                    return xk;

                if (k != 0)
                {
                    sk = xk - xk_1;
                    Log.Add(String.Format("sk:{n{0}}", sk));
                    Matrix<double> m_sk = sk.ToColumnMatrix();
                    Matrix<double> m_sk_t = m_sk.Transpose();

                    yk = Gradient.Calculate(funcND, xk) - Gradient.Calculate(funcND, xk_1);
                    Log.Add(String.Format("yk:{n{0}}", yk));
                    Matrix<double> m_yk = yk.ToColumnMatrix();
                    Matrix<double> m_yk_t = m_yk.Transpose();

                    Hk = Hk - ((Hk * m_yk * m_yk_t * Hk) / (m_yk_t * Hk * m_yk)[0,0]) + ((m_sk * m_sk_t)[0,0] / (m_yk_t * m_sk)
[0,0]);
                    Log.Add(String.Format("Hk:{n{0}}", Hk));
                }

                VectorNMathNet pk = -Hk * gradient;
                Function1D func1D = (double alpha) => { return funcND((xk + pk * alpha)); };
                ak = OneDimensionalMinimization.FindMin(func1D, ak, eps);
                Log.Add(String.Format("ak = {0}", ak));

                // Инкремент
                xk_1 = xk;
                xk = xk + ak * pk;
                k++;
                Log.Add("=====Next Iteration=====");
            }
        }
    }
}
```