Лабораторная работа 3 **Градиентные методы**	**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	
	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).

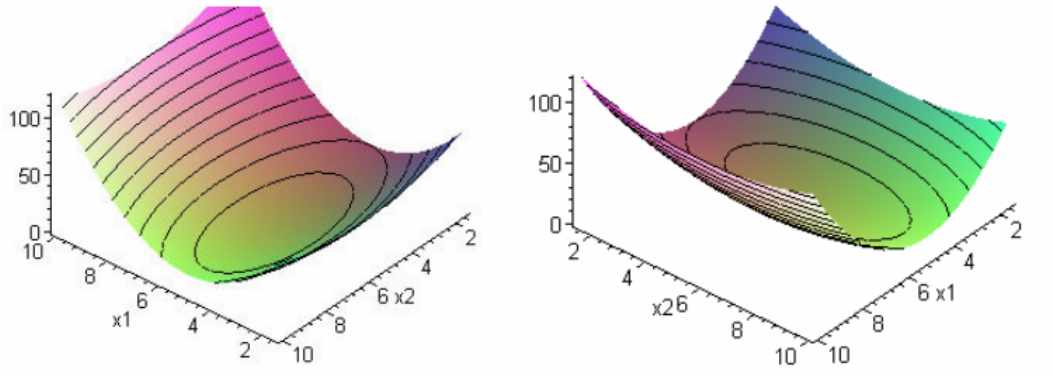


Рис. 6

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

Характеристика	Метод оптимального градиента	Метод переменной метрики
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.

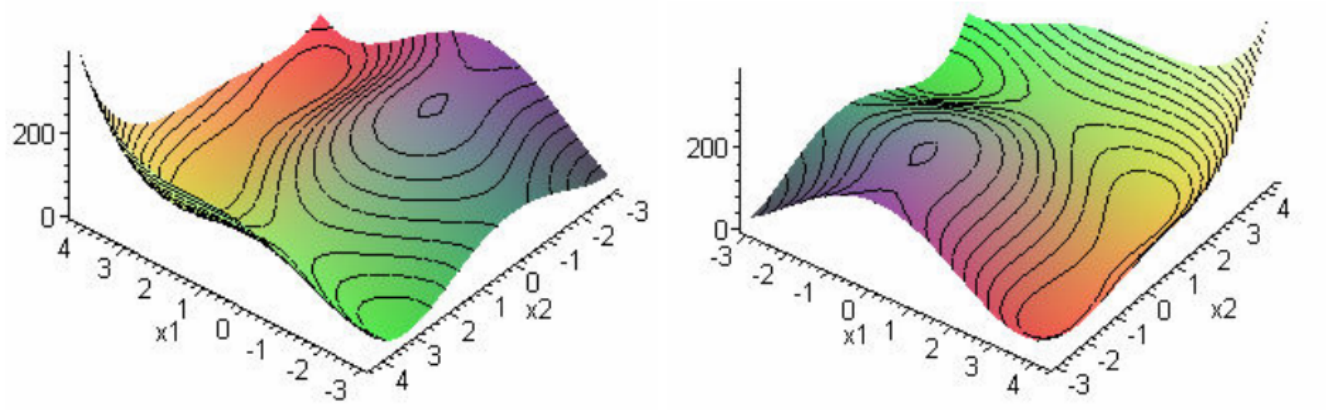


Рис. 7

Тестирующая функция

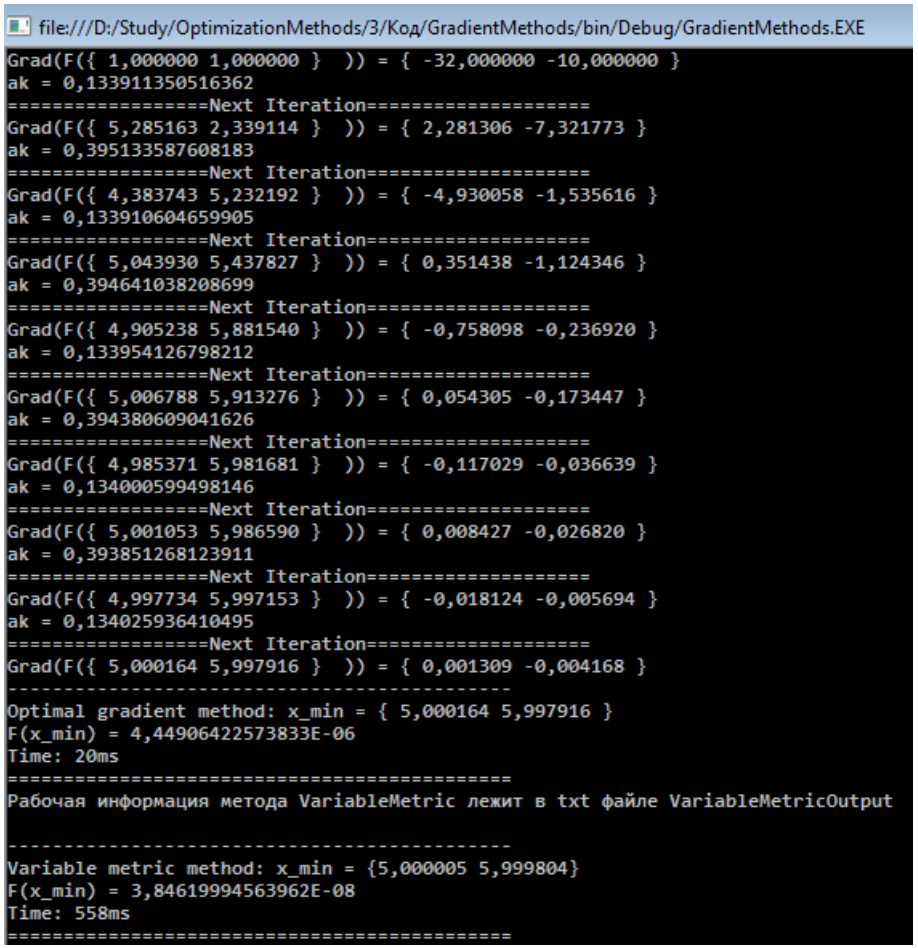
```
// Input data
FunctionNDByAlex funcND = (PointN point) =>
{
        return SMath.Pow(SMath.Pow(point.At(0), 2) + point.At(1) - 11.0, 2.0)
             + SMath.Pow(point.At(0) + SMath.Pow(point.At(1), 2) - 7.0, 2.0);
};
FunctionNDMathNet funcNDMathNet = (VectorNMathNet point) =>
{
        return SMath.Pow(SMath.Pow(point.At(0),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

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

Результаты

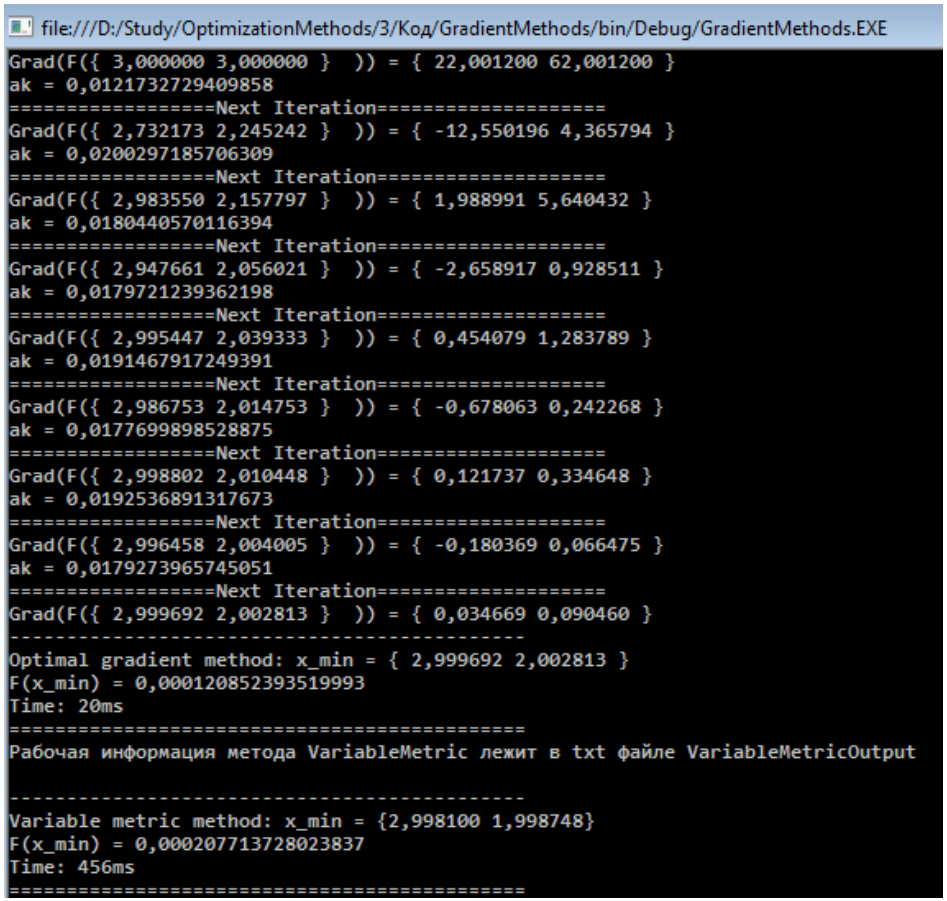
Характеристика	Метод оптимального градиента	Метод переменной метрики
Время выполнения на Core i3	18 ms	287 ms
Относительная ошибка по x[0], %	< 0.01	< 0.01
Относительная ошибка по x[1], %	< 0.01	< 0.01

Результат работы программ на задачах:

2.1



2.1.1



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 gradFxk = Gradient.Calculate(funcND, xk);
        Log.Add(String.Format("Grad(F({0, -23})) = {1, -25}", xk, gradFxk));

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

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

        xk = xk - gradFxk * 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 (nxn)

    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====================");
      }
    }
  }
}
```