УДК 004.021

А.В. Касицкий, П.И. Бидюк

АНАЛИЗ СХОДИМОСТИ МЕТОДА ОПТИМИЗАЦИИ РОЗЕНБРОКА

Аннотация. В работе произведён анализ метода Розенброка, сравнительно методов оптимизации градиентного вида. Предложены две эффективных реализации метода (с учетом результатов полученных в [1]). На примере известных тестовых функций экспериментально получены оценки качества работы данного метода. Сделаны выводы о целесообразности использования метода в различных условиях.

Ключевые слова: метода Розенброка, градиентные методы, дифференцируемость функции.

ВВЕДЕНИЕ

Метод Розенброка [2] является методом многомерной локальной оптимизации, который объединяет в себе преимущества градиентного и покоординатного спуска [7]. Он был опубликован Розенброком в 70х годах. В работе произведён анализ метода, сравнительно методов оптимизации градиентного вида. Предложены две эффективных реализации метода (с учетом результатов полученных в [1]). На примере известных тестовых функций экспериментально получены оценки качества работы данного метода. Сделаны выводы о целесообразности использования метода в различных условиях.

ПОСТАНОВКА ЗАДАЧИ

Пусть целевая функция имеет вид:

$$F(\vec{x}) : R^n \to R, \tag{1}$$

а задача оптимизации задана следующим образом:

$$F(\vec{x}) \to \min_{\vec{x} \in R^n}. \tag{2}$$

Существует много методов решения задачи (2) [3-5, 7]. В данной работе выполнен анализ метода Розенброка, сравнительно градиентных методов [5], т. е. рассмотрен случай дифференцируемости функции (1).

АЛГОРИТМ РОЗЕНБРОКА

Функционирование алгоритма очень похоже на реализацию метода покоординатного спуска. На первой итерации производится по-

иск нулевого порядка (по сути покоординатный спуск) в направлении базисных векторов n-мерного пространства. Главное отличие от метода покоординатного спуска заключается в том, что время от времени (один раз на определенное количество итераций/неудачных итераций) система базисных векторов поворачивается так, что первый вектор направлен вдоль градиента. Далее процесс повторяется уже для новой системы базисных векторов. Создание новой (повернутой) базисной системы векторов, обычно, выполняется процедурой ортогонализации Грамма-Шмидта [8].

РЕАЛИЗАЦИИ АЛГОРИТМА

В данной работе рассмотрены две реализации метода: с делением шага и «делением и умножением» (подробнее в [1]). Алгоритм наискорейшего спуска реализован не был, так как в этом случае для каждой фиксированной базисной системы векторов, производится слишком много вызовов функции, и теряется преимущество, полученное за счет поворота базисной системы. Поворот базисной системы векторов производится один раз для трех неудачных проходов.

В обеих реализациях метода был использован одинаковый размер шага для взятия производной: const cfloat h = 1E-6. Также общей являются функции взятия градиента (градиент считается через центральные производные) и ортогонализация методом Грамма-Шмидта:

```
 vect Grad(vect x){
    vect df;
    for(int i = 0; i < DIM; i++){
        x[i] += h;
        cfloat f1 = F(x);
        x[i] -= 2*h;
        cfloat f2 = F(x);
        x[i] += h;
        df[i] = (f1 - f2)/(2*h);
    }
    return df;
}

void Gram_Schmidt(vect v[]){
    for(int j = 0; j < DIM; ){
        for(int i = 0; i < j; i++)
            v[j] = v[j] - (v[j]*v[i])*v[i];//
||vi|| = 1
        if( v[j].Norm() > EPS )
            v[j++].Normalize();
        else
            swap(v[j],v[DIM]);
    }
}
```

 ISSN 1562-9945

Рассмотрим две конкретных реализации метода Розенброка на языке программирования C++.

Деление шага:

```
void RosenbrokDiv(vect &x){
    vect base[DIM+1];
    for(int i = 0; i < DIM; i++)
        base[i][i] = 1;
    long long itc0 = itc;
    cfloat cur_val = F(x), next_val;
    //we save current and next values
    //to use less function calls
    int it = 0;
    for(cfloat d = INF ; d > EPS ; it++){
    //start with a big step
    //finish when it's realy small
        bool changed = false;
        for(int i = 0; i < DIM; i++){
        //for each base vector
            for(int s = -1 ; s <= 1 ; s+=2){
            //for both directions
                x = x + base[i] * d * s;
                //try to move
                next_val = F(x);
                if( next_val < cur_val ){
                //if it's better
                    changed = true;
                    cur_val = next_val;
                    break;//don't try another direction
                }else
                    x = x - base[i] * d * s;
                    //move back
            }
        }
        if( !changed ){
        //if we're not able to move with current step
            //make it smaller
            if( it % 3 == 0 ){
            //if it happened few times already
                base[DIM] = Grad(x);
                base[DIM].Normalize();
                swap(base[0],base[DIM]);
                Gram_Schmidt(base);
                //turn the base vectors
            }
        }

    }
    cout<<"function F was executed "<<itc-itc0<<"
times"<<endl;
}
```

Деление и умножение шага:

```
void RosenbrokDivMul(vect &x){
    vect base[DIM+1];
    for(int i = 0; i < DIM; i++)
        base[i][i] = 1;
    long long itc0 = itc;
    cfloat cur_val = F(x), next_val;
    //we save current and next values
    //to use less function calls
    int it = 0;
    for(cfloat d = INF ; d > EPS ; it++){
    //start with a big step
    //finish when it's realy small
        bool changed = false;
        for(int i = 0; i < DIM; i++){
        //for each base vector
            for(int s = -1 ; s <= 1 ; s+=2){
            //for both directions
                x = x + base[i] * d * s;
                //try to move
                next_val = F(x);
                if( next_val < cur_val ){
                //if it's better
                    changed = true;
                    cur_val = next_val;
                    break;//don't try another direction
                }else
                    x = x - base[i] * d * s;
                    //move back
            }
        }
        if( !changed ){
        //if we're not able to move with current step
            d /= 2;
            //make it smaller
            if( it % 3 == 0 ){
            //if it happened few times already
                base[DIM] = Grad(x);
                base[DIM].Normalize();
                swap(base[0],base[DIM]);
                Gram_Schmidt(base);
                //turn the base vectors
            }
        }else
            d *= 2;
            //make it bigger

    }
    cout<<"function F was executed "<<itc-itc0<<"
times"<<endl;
}
```

В работе экспериментально исследованы «особенности поведения» реализованных методов на примере функций, описанных в [1].

 ISSN 1562-9945

Это так называемая «хорошая» функция, функция Розенброка и «плохая» функция. Все тесты полностью идентичны соответствующим тестам из [1]. Для сравнения приведены результаты некоторых реализаций (всех кроме наискорейшего спуска), исследованных в [1].

«ХОРОШАЯ» ФУНКЦИЯ

$F(x,y) = x^2 + y^2 + 3x - 4y + 2$ — гладкая, выпуклая функция. Глобальный минимум существует, единственный и достигается в точке $x^* = -1.5, y^* = 2$ (рис. 1).

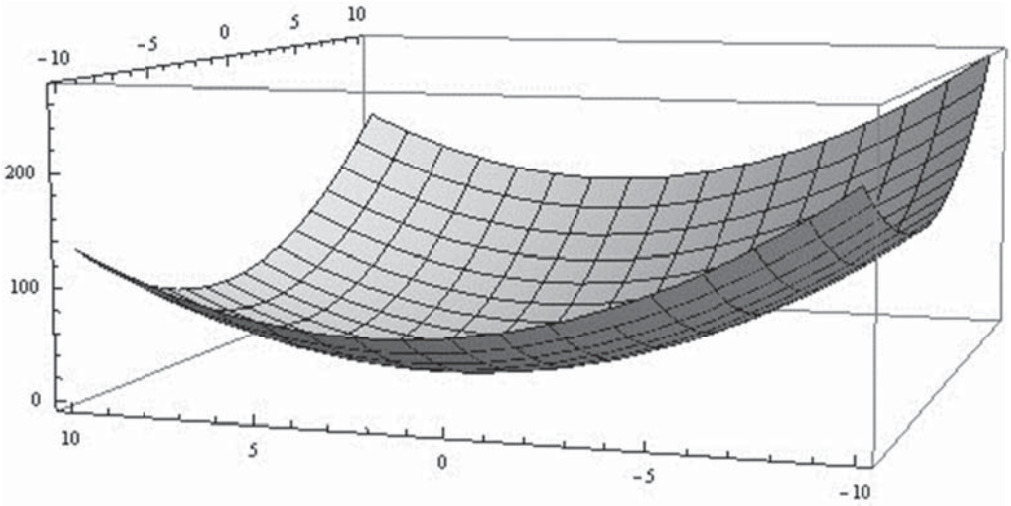

Рисунок 1 – Глобальный минимум «хорошей функции

Тест 1: хорошая функция.

Начальное приближение $x_0 = 123, y_0 = -321$, бесконечность: INF =1E9, точность: EPS = 1E-9.

Таблица 1

Результаты поиска для хорошей функции

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	3.8e-10	1.5e-10	0.0e+0	0.0e+0	334
CoordinateDivMul	3.8e-10	1.5e-10	0.0e+0	0.0e+0	450
CoordinateSteepest	3.8e-10	1.5e-10	0.0e+0	0.0e+0	702
GradientDiv	1.8e-8	7.4e-9	3.5e-16	8.2e-17	985
GradientDivNorm	1.6e-9	6.5e-10	2.6e-18	6.1e-19	391
GradientDivMulNorm	1.6e-9	6.5e-10	2.6e-18	6.1e-19	481
GradientSteepest	1.0e-10	4.0e-11	0.0e+0	0.0e+0	356
RosenbrokDiv	1.6e-9	6.5e-10	2.6e-18	6.1e-19	381
RosenbrokDivMul	1.6e-9	6.5e-10	2.6e-18	6.1e-19	454

В первом столбце таблицы 1 название алгоритма, во втором/третьем – абсолютная/относительная погрешность нахождения x*, в четвертом/пятом столбцах – абсолютная/относительная погрешность нахождения F(x*), в последнем столбце – количество вызовов целевой функции. Как видно из таблицы 1, метод Розенброка дал такие же результаты, как и методы, исследованные в [1].

ФУНКЦИЯ РОЗЕНБРОКА

$F(x,y) = (1-x)^2 + 100(y-x^2)^2$ — гладкая, невыпуклая функция. Глобальный минимум существует, единственный и достигается в точке $x^* = 1, y^* = 1$ (рис. 2).

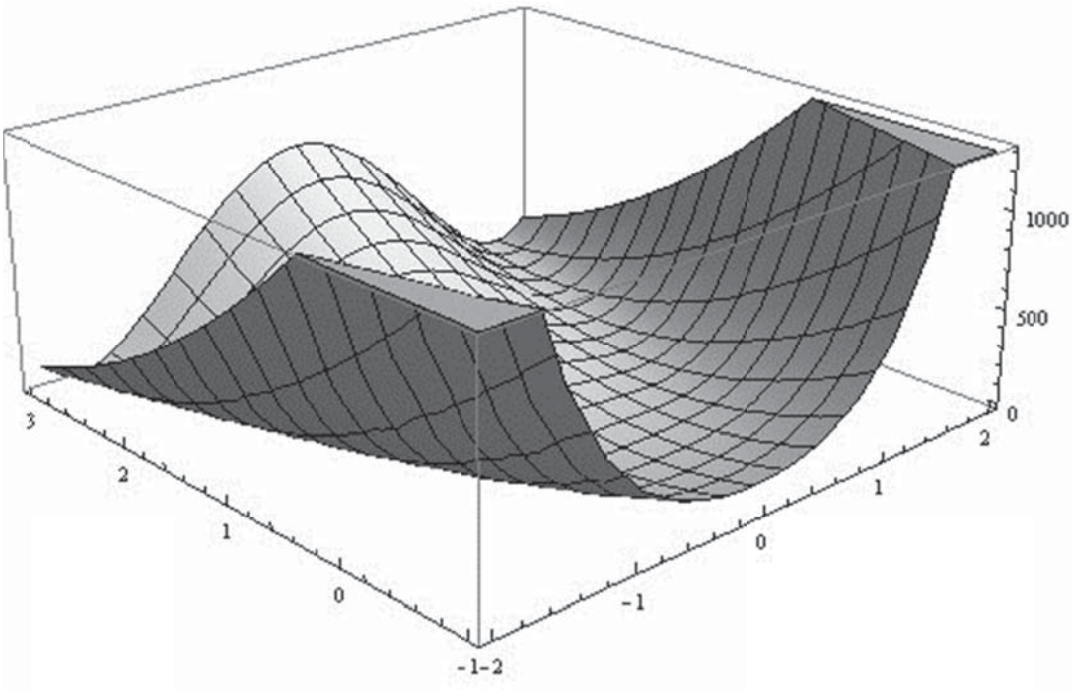

Рисунок 2 – Функция Розенброка

Таблица 2

Результаты поиска для функции Розенброка

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	9.7e-7	6.9e-7	1.9e-13	n/a	1,592,528,468
CoordinateDivMul	9.7e-7	6.8e-7	1.8e-13	n/a	163,948,785
CoordinateSteepest	7.6e-7	5.4e-7	1.1e-13	n/a	1,318,302
GradientDiv	1.1e-2	7.9e-3	2.5e-5	n/a	232,200,030
GradientDivNorm	2.4e-7	1.7e-7	1.1e-14	n/a	84,481
GradientDivMulNorm	5.2e-8	3.6e-8	5.4e-16	n/a	182,591
RosenbrokDiv	1.2e-7	8.9e-8	3.4e-15	n/a	49,728,621
RosenbrokDivMul	2.9e-7	2.0e-7	1.7e-14	n/a	95,423

 ISSN 1562-9945

Тест 1: функция Розенброка.

Начальное приближение $x_0 = 123, y_0 = -321$, бесконечность — INF =1E9, точность — EPS = 1E-9.

Как видно, метод Розенброка с делением и умножением шага привел к намного лучшему результату, чем реализация с делением шага. Метод с делением и умножением, обогнал все методы кроме градиентного с делением шага и нормировкой градиента.

Тест 2: функция Розенброка.

Начальное приближение $x_0 = 123, y_0 = -321$, бесконечность — INF =1E9, точность — EPS = 1E-12.

Таблица 3

Результаты поиска экстремума для функции Розенброка

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	4.5e-10	3.2e-10	4.1e-20	n/a	1,592,539,501
CoordinateDivMul	4.5e-10	3.2e-10	4.1e-20	n/a	163,968,322
CoordinateSteepest	1.1e-9	8.0e-10	2.5e-19	n/a	2,323,658
GradientDiv	1.1e-5	7.9e-6	2.5e-11	n/a	621,017,650
GradientDivNorm	1.5e-9	1.0e-9	4.6e-19	n/a	118,041
GradientDivMulNorm	1.4e-9	1.0e-9	4.1e-19	n/a	228,111
RosenbrokDiv	7.1e-11	5.0e-11	1.0e-21	n/a	49,731,039
RosenbrokDivMul	1.0e-10	7.5e-11	2.2e-21	n/a	99,695

Для методов Розенброка количество итераций изменилось намного меньше, чем у градиентных. В то же время увеличение точности оказалось у них самым большим (табл. 3). Можно понять, что большую часть «времени», они тратят далеко от точки минимума (аналогично координатным методам с дробным шагом, как было показано в [1]). На данном тесте метод Розенброка с делением и умножением шага обогнал все остальные методы (хотя преимущество по сравнению с градиентным методом с делением шага и невелик).

Таблица 4

Результаты поиска экстремума для функции Розенброка

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	8.9e-10	6.3e-10	1.6e-19	n/a	1,867,763
CoordinateDivMul	8.9e-10	6.3e-10	1.6e-19	n/a	884,977
CoordinateSteepest	1.0e-9	7.4e-10	2.2e-19	n/a	2,324,240
GradientDiv	8.8e-8	6.2e-8	1.5e-15	n/a	6,987,095
GradientDivNorm	1.5e-9	1.0e-9	4.6e-19	n/a	120,186
GradientDivMulNorm	1.3e-9	9.8e-10	3.8e-19	n/a	230,841
RosenbrokDiv	8.4e-10	5.9e-10	1.4e-19	n/a	44,898
RosenbrokDivMul	5.1e-10	3.6e-10	5.3e-20	n/a	27,077

Тест 3: функция Розенброка.

Начальное приближение $x_0 = 12, y_0 = -32$, бесконечность — INF =1E9, точность — EPS = 1E-12.

Если взяв начальную точку ближе к точке минимума, то методы Розенброка дают лучшие результаты по сравнению со всеми остальными методами с существенным перевесом (табл. 4). Теперь выберем точку еще ближе к точке минимума.

Тест 4: функция Розенброка.

Начальное приближение $x_0 = 3, y_0 = -3$, бесконечность — INF =1E9, точность — EPS = 1E-12.

Таблица 5

Результаты поска экстремума для функции Розенброка

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	1.0e-9	7.2e-10	2.1e-19	n/a	59,634
CoordinateDivMul	1.0e-9	7.2e-10	2.0e-19	n/a	94,006
CoordinateSteepest	1.1e-9	7.9e-10	2.5e-19	n/a	2,323,968
GradientDiv	1.1e-8	8.1e-9	2.6e-17	n/a	987,780
GradientDivNorm	1.5e-9	1.0e-9	4.6e-19	n/a	117,206
GradientDivMulNorm	1.3e-9	9.8e-10	3.8e-19	n/a	225,601
RosenbrokDiv	6.5e-10	4.6e-10	8.4e-20	n/a	38,792
RosenbrokDivMul	4.6e-10	3.2e-10	4.3e-20	n/a	5,434

Как видно (табл. 5), время работы методов Розенброка снова значительно уменьшилось.

На тех тестах, на которых координатные методы сильно проигрывали, методы Розенброка справлялись на уровне с градиентными методами. На тех тестах, где координатные методы выигрывали у градиентных, методы Розенброка выиграли у градиентных методов еще больше. На всех тестах метод с делением и умножением шага работал лучше.

«ПЛОХАЯ» ФУНКЦИЯ

$F(x, y) = x^2 + 1000\dfrac{y^2}{x^2 + 0.01}$ — гладкая, не выпуклая функция Глобальный минимум существует, единственный и достигается в точке $x^* = 0, y^* = 0$ (рис. 3).

 ISSN 1562-9945

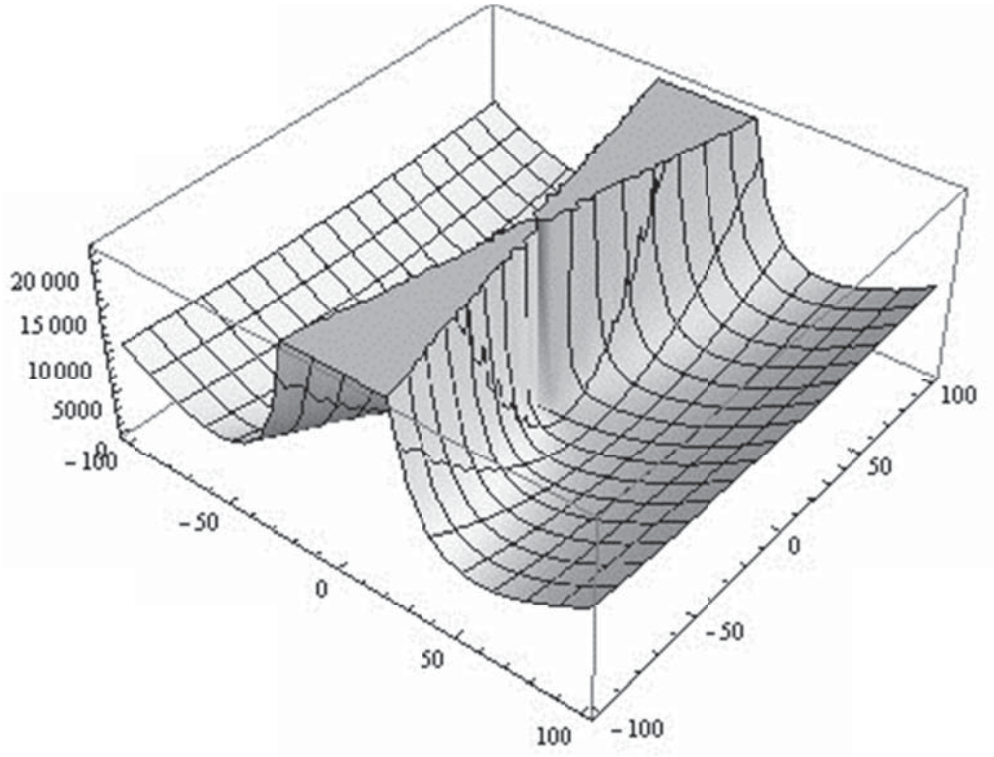

Рисунок 3 – Вид «плохой» функции

Тест 1: «плохая» функция.

Начальное приближение $x_0 = 123, y_0 = -321$, бесконечность — INF =1E9, точность — EPS = 1E-9.

Таблица 6

Результаты поиска для «плохой» функции

Algorithm	|x-x*|	|x-x*|/|x*|	|F(x)-F(x*)|	|F(x)-F(x*)|/|F(x*)|	F() calls#
CoordinateDiv	8.8e-10	n/a	4.7e-14	n/a	891
CoordinateDivMul	8.8e-10	n/a	4.7e-14	n/a	700
CoordinateSteepest	4.0e-10	n/a	6.6e-15	n/a	940
GradientDiv	7.0e-5	n/a	4.9e-9	n/a	2,589,625
GradientDivNorm	3.4e-6	n/a	1.2e-11	n/a	4,492,921
GradientDivMulNorm	1.8e-8	n/a	3.5e-16	n/a	9,867,881
RosenbrokDiv	5.9e-5	n/a	3.5e-9	n/a	735,927
RosenbrokDivMul	1.2e-6	n/a	1.6e-12	n/a	9,560

Как видно из таблицы 6, координатные методы справились с задачей лучше других, градиентные хуже других. Методы Розенброка оказались посредине. Это в очередной раз доказывает то, что они сочетают в себе свойства обоих методов. Необходимо заметить, что метод Розенброка с делением и умножением шага на порядок медленнее координатных методов, и на три порядка быстрее, чем градиентные.

ВЫВОДЫ

Рассмотрены две реализации метода Розенброка. Сходимость этих методов протестирована на трех функциях: «хорошей» (выпуклой), функции Розенброка с разными начальными приближениями, и «плохой» функции. Результаты тестирования сравнены с результатами, полученными в [1]. В целом можно сделать следующие выводы: (1) для «хороших» функций все методы работают достаточно хорошо; (2) реализация метода Розенброка с делением и умножением шага в среднем работает значительно лучше; (3) на всех предложенных функциях, метод Розенброка с делением и умножением шага работал достаточно быстро; (4) метод Розенброка является, в некотором смысле, рациональным компромиссом между координатным и градиентным методами; (5) метод Розенброка является эффективным и в отличие от координатного и градиентного методов достаточно универсальным.

ЛИТЕРАТУРА

1. Касицкий А. Сравнение градиентных методов / Сборник «Системные науки и кибернетика». Киев, НТУУ «КПИ», 2010. – С. 75 – 84.
2. Rosenbrock H. H. An automatic Method for findong the greatest or least Value of a Function // Computer Journal, 1960, No. 3, pp. 175-184.
3. Химмельблау Д. Прикладное нелинейное программирование. М.: Мир, 1974. – 536 с.
4. Полак Э. Численные методы оптимизации. Единый подход. М.: Мир, 1974. -376 с.
5. Метод градиентного спуска. http://en.wikipedia.org/wiki/Gradient_descent
6. Функция Розенброка. http://en.wikipedia.org/wiki/Rosenbrock_function
7. Метод покоординатного спуска. – http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%.
8. Gram-Schmidt Orthonormalization. – http://mathworld.wolfram.com/Gram-SchmidtOrthonormalization.html.

 ISSN 1562-9945