Санкт-Петербургский государственный университет

**Курсовая работа**

по дисциплине «Численные методы»

на тему: «Методы наискорейшего спуска»

Студент: Марусина Анастасия Петровна

Группа: 209

Преподаватель: Перегудин Сергей Иванович

Методы наискорейшего спуска

Поскольку сходящаяся минимизирующая последовательность позволяет найти точку минимума рассматриваемой функции с произвольной наперед заданной точностью, то следует рассмотреть вопрос о построении такой последовательности. Поставим следующую задачу: имея точку построить точку такую, чтобы выполнялось соотношение:

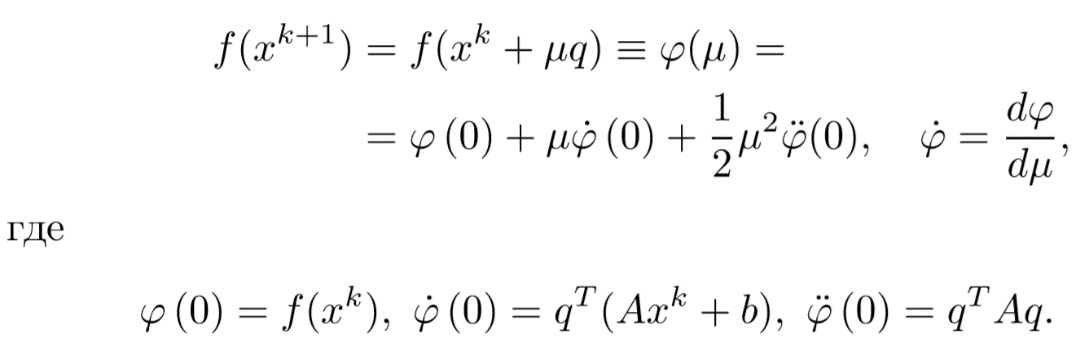
(1)

Будем искать точку в следующем виде:

(2)

где q – заданный вектор из , называемый направлением спуска, а µ – искомый параметр, называемый обычно шагом метода в направлении спуска.

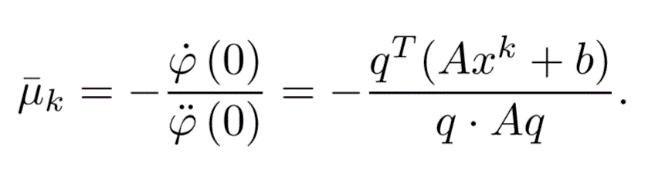
Параметр µ (шаг метода) в формуле (2) определим из условия минимума функции f(·) на прямой (2). Имеем:



(3)

(4)

Поскольку является квадратным трёхчленом относительно µ с положительным старшим коэффициентом, то у этого трёхчлена существует точка минимума , которая может быть найдена из необходимого условия экстремума , откуда и получаем:

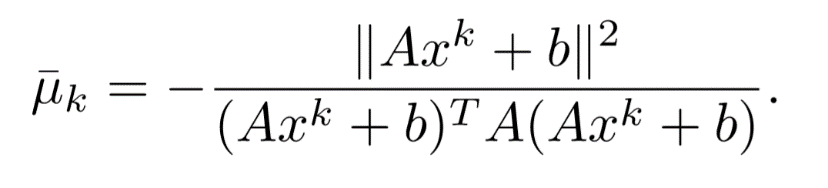


(5)

Очевидно, что если в формуле (2) положить , то построенная по формуле (2) точка будет удовлетворять условию (1), причём там будет строгое неравенство в случае . Продолжая указанные построения, получим последовательность {}, которую естественно назвать последовательностью убывания для функции f(·).

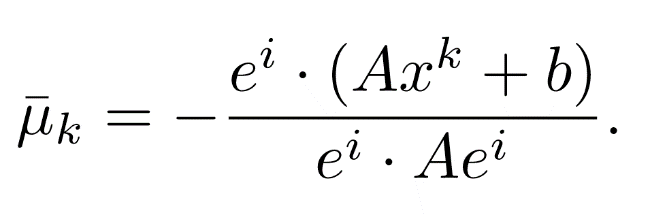
Метод наискорейшего градиентного спуска (МНГС)

Если в формулах (2) считать, что (то есть вектор q является вектором градиента функции f(·) в точке , то соответствующий метод построения последовательности {} называют градиентным методом. Если к тому же шаг метода µ в формуле (2) выбирается по формуле (5), то такой метод называют (одношаговым) методом наискорейшего градиентного спуска (МНГС). В этом случае формула (5) принимает вид



Метод покоординатного спуска (МПС)

В случае выбора направлений спуска q в формуле (2) на каждом шаге в , где – i-ый орт пространства , метод носит название метода покоординат- ного спуска. При выборе шага метода по формуле (5) его называют методом наискорейшего покоординатного спуска (МНПС). В этом случае формула (5) принимает вид



Реализация метода наискорейшего градиентного спуска на языке C++ используя интегрированную среду разработки Visual Studio 2013:

#include "stdafx.h"

#include <iostream>

using namespace std;

float CalculateD(float derivative[4], float X[3])

{

return derivative[0] \* X[0] + derivative[1] \* X[1] + derivative[2] \* X[2] + derivative[3];

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

float expression[10] = {

//x^2 y^2 z^2 xy xz yz x y z

2.0, 3.8, 4.8, 1, 1, -1, 1, -2, 3, 8

};

float eps = 0.01;

float derivativeX1[4];

derivativeX1[0] = expression[0] \* 2; //X

derivativeX1[1] = expression[3]; //Y

derivativeX1[2] = expression[4]; //Z

derivativeX1[3] = expression[6]; //0

float derivativeX2[4];

derivativeX2[0] = expression[3]; //X

derivativeX2[1] = expression[1] \* 2; //Y

derivativeX2[2] = expression[5]; //Z

derivativeX2[3] = expression[7]; //0

float derivativeX3[4];

derivativeX3[0] = expression[4]; //X

derivativeX3[1] = expression[5]; //Y

derivativeX3[2] = expression[2] \* 2; //Z

derivativeX3[3] = expression[8]; //0

float X0[3] = {

1, 1, 1

};

float D0[3];

D0[0] = CalculateD(derivativeX1, X0);

D0[1] = CalculateD(derivativeX2, X0);

D0[2] = CalculateD(derivativeX3, X0);

float X[3];

float p = 1;

int counter = 0;

while (p > eps)

{

counter++;

float L0 = (expression[0] \* D0[0] \* D0[0]

+ expression[1] \* D0[1] \* D0[1]

+ expression[2] \* D0[2] \* D0[2]

+ expression[3] \* D0[0] \* D0[1]

+ expression[4] \* D0[0] \* D0[2]

+ expression[5] \* D0[1] \* D0[2]) \* 2; // L

float L1 = expression[0] \* 2 \* X0[0] \* D0[0]

+ expression[1] \* 2 \* X0[1] \* D0[1]

+ expression[2] \* 2 \* X0[2] \* D0[2]

+ expression[3] \* X0[0] \* D0[1]

+ expression[3] \* X0[1] \* D0[0]

+ expression[4] \* X0[0] \* D0[2]

+ expression[4] \* X0[2] \* D0[0]

+ expression[5] \* X0[1] \* D0[2]

+ expression[5] \* X0[2] \* D0[1]

+ expression[6] \* D0[0]

+ expression[7] \* D0[1]

+ expression[8] \* D0[2]; // 0

float L = -1 \* L1 / L0; // alpha

X[0] = X0[0] + L\*D0[0];

X[1] = X0[1] + L\*D0[1];

X[2] = X0[2] + L\*D0[2];

//check stop

float P[3];

P[0] = CalculateD(derivativeX1, X);

P[1] = CalculateD(derivativeX2, X);

P[2] = CalculateD(derivativeX3, X);

p = sqrt(P[0] \* P[0] + P[1] \* P[1] + P[2] \* P[2]);

cout << "Continue " << p << endl;

//D01

float D01[3];

D01[0] = P[0];

D01[1] = P[1];

D01[2] = P[2];

// repeat

D0[0] = D01[0];

D0[1] = D01[1];

D0[2] = D01[2];

X0[0] = X[0];

X0[1] = X[1];

X0[2] = X[2];

}

cout << "Succes!" << endl;

cout << "Count: " << counter << endl;

cout << "X" << ": " << X[0] << endl;

cout << "Y" << ": " << X[1] << endl;

cout << "Z" << ": " << X[2] << endl;

system("pause");

return 0;

}

Реализация метода покоординатного спуска на языке C++ используя интегрированную среду разработки Visual Studio 2013:

#include "stdafx.h"

#include <iostream>

using namespace std;

float CalculateD(float derivative[4], float X[3])

{

return derivative[0] \* X[0] + derivative[1] \* X[1] + derivative[2] \* X[2] + derivative[3];

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

float expression[10] = {

//x^2 y^2 z^2 xy xz yz x y z

2.0, 3.8, 4.8, 1, 1, -1, 1, -2, 3, 8

};

float eps = 0.01;

float derivativeX1[4];

derivativeX1[0] = expression[0] \* 2; //X

derivativeX1[1] = expression[3]; //Y

derivativeX1[2] = expression[4]; //Z

derivativeX1[3] = expression[6]; //0

float derivativeX2[4];

derivativeX2[0] = expression[3]; //X

derivativeX2[1] = expression[1] \* 2; //Y

derivativeX2[2] = expression[5]; //Z

derivativeX2[3] = expression[7]; //0

float derivativeX3[4];

derivativeX3[0] = expression[4]; //X

derivativeX3[1] = expression[5]; //Y

derivativeX3[2] = expression[2] \* 2; //Z

derivativeX3[3] = expression[8]; //0

float X0[3] = {

1, 1, 1

};

float X[3];

float p = 1;

int counter = 0;

while (p > eps)

{

counter++;

X[0] = (-1 \* derivativeX1[1] \* X0[1] - derivativeX1[2] \* X0[2] - derivativeX1[3]) / derivativeX1[0];

X[1] = (-1 \* derivativeX2[0] \* X[0] - derivativeX2[2] \* X0[2] - derivativeX2[3]) / derivativeX2[1];

X[2] = (-1 \* derivativeX3[0] \* X[0] - derivativeX3[1] \* X[1] - derivativeX3[3]) / derivativeX3[2];

//check stop

float P[3];

P[0] = X[0] - X0[0];

P[1] = X[1] - X0[1];

P[2] = X[2] - X0[2];

//norma

p = sqrt(P[0] \* P[0] + P[1] \* P[1] + P[2] \* P[2]);

cout << "Continue " << p << endl;

X0[0] = X[0];

X0[1] = X[1];

X0[2] = X[2];

}

cout << "Succes!" << endl;

cout << "Count: " << counter << endl;

cout << "X" << ": " << X[0] << endl;

cout << "Y" << ": " << X[1] << endl;

cout << "Z" << ": " << X[2] << endl;

system("pause");

return 0;

}