|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | |  | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  Федеральное государственное бюджетное образовательное учреждение  высшего образования  “Московский авиационный институт”   1. (национальный исследовательский университет) | |
| Институт №3 «Системы управления, информатика и электроэнергетика» |
| Кафедра 307 «Цифровые технологии и информационные системы» |

|  |
| --- |
| Курсовая работа  По дисциплине:  «Технологии программирования»  Тема: **“Метод Розенброка с дискретным шагом”** |
|  |
|  |

Выполнил

Ст. группы М3О-412Б-18 Акимов В.Н.

Проверил:

Профессор Татарникова Е.М.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

МАИ2021

**Постановка задачи**

Затраты топлива БПЛА при полете над сектором местности на фиксированной высоте определяются следующим соотношением:

F1(x) = 9X12 + 16X2 2 - 90X1- 128X2,

где X1, X2 – широта и долгота.

Коэффициент искажения качества снимков, производимых БПЛА при изменении траектории полета определяется следующим соотношением:

F2(x) = X12 + 2X1 X2  + 2X22 + X32 - X2X3 +X1 + 3X2 - X3

где X1 - скорость полета, X2 – угловая скорость в оси тангажа, X3 – высота полета.

Провести:

1. Найти методом Розенброка c дискретным шагом экстремумы функций F1(x) и F2(x) при различных значениях начальной точки;
2. Определение экстремума заданных функций при различных значениях точности

( ;

1. На каждой итерации метода предусмотреть вывод следующей информации:

* номера текущей итерации;
* координаты текущей точки;
* значение функции в текущей точке;
* на каждой итерации: номер переменной (j), допустимое направление dj, величину шага λj по направлению j, координату полученной точки (yj+1) и т.д.

Пример вывода результатов вычислений по методу Розенброка с дискретным шагом:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K | Xk  F(Xk) | j | yj  F(yj) | j | dj | yjjdj  F(yjjdj) |
| 1 | (0.00,3.00)  52.00 | 1    2  1  2  . . . . | (0.00, 3.00)  52.00  (0.10, 3.00)  47.84  (0.10, 3.00)  47.84  (0.30, 3.00)  40.84  . . . . . . | 0.10  0.10  0.20  - 0.05 | (1.0, 0.0)  (0.0, 1.0)  (1.0, 0.0)  (0.0, 1.0) | (0.10, 3.00)  47.84  (0.10, 3.10)  50.24  (0.30, 3.00)  40.84  (0.30, 2.95)  39.71 |
| 2 | . . . . . . |  |  |  |  |  |
| . . |  |  |  |  |  |  |
| N |  |  |  |  |  |  |

1. В конце вычислений выводить следующую информацию:

* оптимальное значение аргумента;
* оптимальное значение функции;
* количество итераций метода для достижения оптимальной точки.

1. Построить график и показать на нем:

* линии уровня функции F1(x);
* точки, получаемые на каждой итерации, начиная от начальной и заканчивая конечной, с изображением вектора направления от каждой предыдущей точки до последующей.

6. Сделать выводы по результатам исследований.

**Формулировка метода Розенброка с дискретным шагом**

Задать начальные параметры:

> 0 – точность;

> 1 – коэффициент растяжения;

(-1, 0) – коэффициент сжатия,

– начальные длины шагов вдоль каждого направления;

– начальная точка.

Положить

Шаг 1.

Если , то шаг считается успешным и он осуществляется: . При этом .

Если же , то шаг считается неудачным: и .

Шаг 2.

Если , то положить и перейти на шаг 1.

Шаг 3.

Если , т.е. спуск хотя бы по одному из направлений оказался удачным, то и перейти на шаг 1.

Шаг 4.

Если , т.е. каждый из последних спусков по направлению был неудачным) и , то перейти на шаг 5.

Если же последнее условие не выполняется (т.е. ), то при (конец работы алгоритма), иначе , , перейти на шаг 1.

Шаг 5.

Положить . Если , то (конец работы алгоритма), иначе вычислить из соотношения и построить новые направления , на основе следующих соотношений:

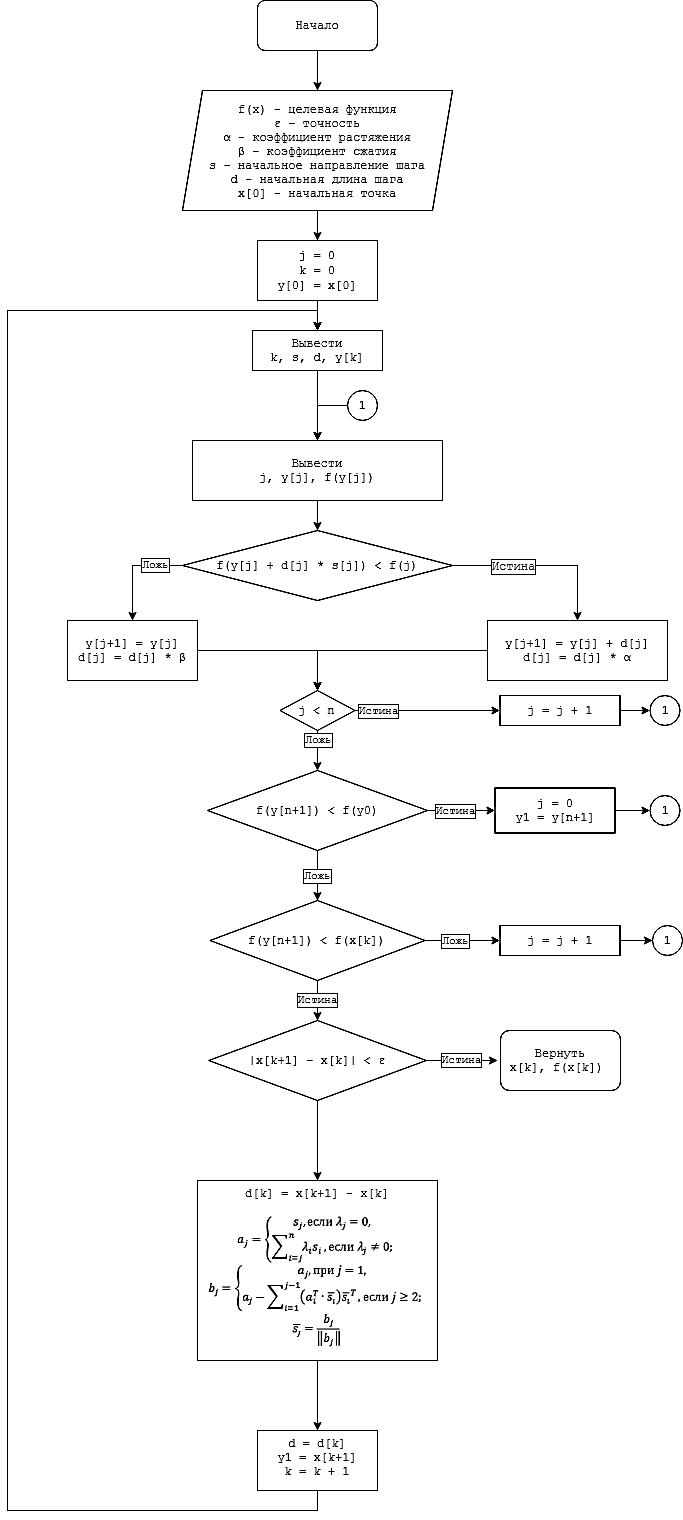
Шаг 6.

Положить: для . Перейти на шаг 1.

**Ход выполнения работы**

Перед разработкой программы процедурным подходом была составлена блок-схема алгоритма оптимизации, по которой был составлен псевдокод. Также были рассчитаны первые несколько шагов метода, для дальнейшего тестирования программы.

*Рисунок 1 - Блок-схема алгоритма Розенброка с дискретным шагом.*



Псевдокод алгоритма Розенброка с дискретным шагом:

START

INPUT ε

INPUT delta

INPUT α

INPUT β

INPUT x1

y := x1

n := 2

k := 1

j := 0

s := [[1, 0], [0, 1]]

WHILE k < 10

PRINT k

PRINT y

value := function(y)

PRINT value

WHILE j < n

y\_with\_step := y + delta[j] + s[j]

PRINT y\_with\_step

stepvalue = function(y\_with\_step)

IF step\_value < value THEN

delta[j] := delta[j] \* a

y := y\_with\_step

last\_successful\_point := y\_with\_step

ELSE

delta[j] := delta[j] \* b

ENDIF

ENDIF

IF y = y\_with\_step

y := last\_successful\_point

j := 0

CONTINUE

ELSEIF |funtction(y) – funtction(y\_with\_step)| < ε

BREAK

ENDIF

FOR j = 0, ..., n

delta[j] := last\_successful\_point[j] – y[j]

ENDFOR

A := []

FOR j = 0, ..., n

FOR i = j, ..., n

A = A + delta[i]\*s[i]

ENDFOR

ENDFOR

s[0] = A[0] / ||A[0]||

B := []

FOR j = 0, ..., n

B\_sum := []

FOR i = 0, ..., j

B\_sum := B\_sum + A[j]T \* s[i] \* s[i]T

ENDFOR

B := A[j] – B\_sumT

S[j]= B / ||B||

ENDFOR

j := 0

ENDWHILE

PRINT k

PRINT y

PRINT function(y)

END

Для ручного счета минимума функции F1 зададим начальные параметры:

ε = 0.01

α= 3.0

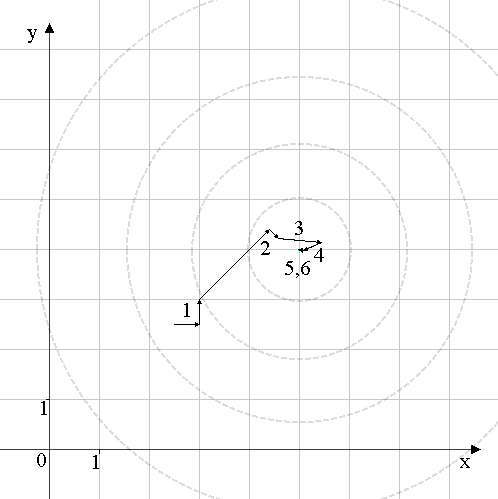
β = -0.5

s =

delta = (2.5, 2.5)

*Таблица 1 - Ручное вычисление итераций для функции F1.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K | Xk  F(Xk) | j | yj  F(yj) | j | dj | yjjdj  F(yjjdj) |
| 1 | (2.50,2.50)  -388.75 | 0    1 | (2.50, 2.50)  -388.75  (3.00, 2.50)  -409.00 | 0.5  0.5 | (1.0, 0.0)  (0.0, 1.0) | (3.00, 2.50)  -409.00  (3.00, 3.00)  -429.00 |
| 2 | (3.00, 2.50)  -429.0 | 0  1  0  1 | (3.00, 2.50)  -429.0  (3.35, 3.35)  -449.92  (3.35, 3.35)  449.92  (4.41, 4.41)  -475.17 | 0.5  0.5  1.5  -0.25 | (0.70, 0.70)  (-0.70, 0.70)  (0.70, 0.70)  (-0.70, 0.70) | (3.35, 3.35)  -449.92  (3.00, 3.71)  -448.91  (4.41, 4.41)  -475.17  (4.59, 4.24)  -478.59 |
| 3 | (4.59, 4.24)  -478.59 | 0  1  0  1 | (4.59, 4.24)  -478.59  (4.59, 4.24)  -478.59  (4.59, 4.24)  -478.59  (4.59, 4.24)  -478.59 | 2.09  1.74  -1.05  -0.87 | (0.09, 0.99)  (-0.99, 0.09)  (0.09, 0.99)  (-0.99, 0.09) | (4.78, 6.32)  -394.49  (2.86, 4.40)  -478.59  (4.49, 3.19)  -478.59  (5.56, 4.16)  -478.59 |
| 4 | . . . . . . |  |  |  |  |  |
| . . |  |  |  |  |  |  |
| 7 | (4.99, 4.00)  -480.99 | 1  2 | (4.99, 4.00)  -480.99  (5.00, 4.00)  -481.00 | -0.02  -0.01 | (-0.90, -0.43)  (0.43, -0.90) | (5.00, 4.00)  -481.00  (5.00, 4.00)  -481.00 |



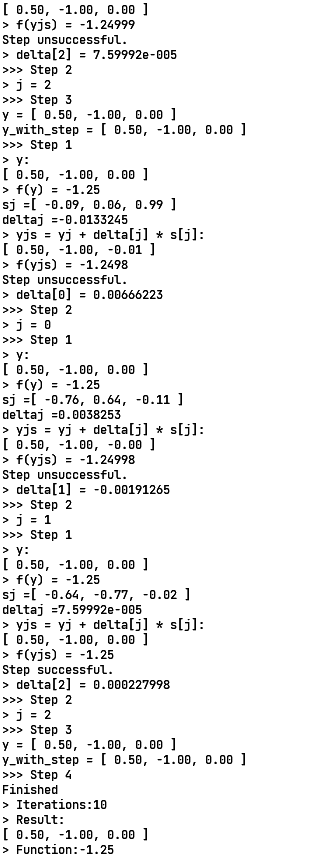
*Рисунок 2 – График шагов итераций (цифрами указаны номера итераций).*

Пример выполнения программы, написанной с применением процедурного подхода, для функции F1. Исходные значения параметров:

|  |  |
| --- | --- |
| *Рисунок 3 - Начало выполнения процедурной программы для функции F1.* | *Рисунок 4 - Завершение выполнения процедурной программы для функции F1.* |

Пример выполнения программы, написанной с применением процедурного подхода, для функции F2. Исходные значения параметров:

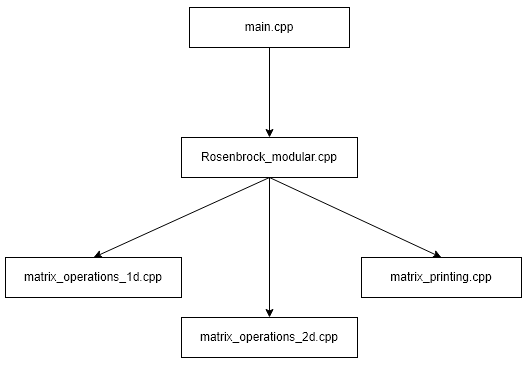
*Рисунок 5 – Итог выполнения процедурной программы для функции F2.*

****

При применении модульного подхода, алгоритм программы было принято разбить на 4 составляющие:

1. Модуль “Rosenbock\_modular” содержащий функцию с основным блоком кода выполняющим итерации алгоритма оптимизации. Он содержит 3 зависимости к модулям, содержащим операции для работы с матрицами.
2. Модуль ”matrix\_operations\_2d” содержащий функции для работы с одномерными массивами: суммирование, разность, проверка на эквивалентность, умножение на скаляр, нахождение нормы, перевод в двумерную матрицу.
3. Модуль “matrix\_operations\_1d” содержащий функции для выполнения операций с двумерными матрицами: суммирование, разность, транспонирование и умножение.
4. Модуль “matrix\_printing” содержащий функции для выведения одномерных и двумерных массивов в консоль.

*Рисунок 6 - Древовидная схема для программы, написанной с применением модульного подхода.*

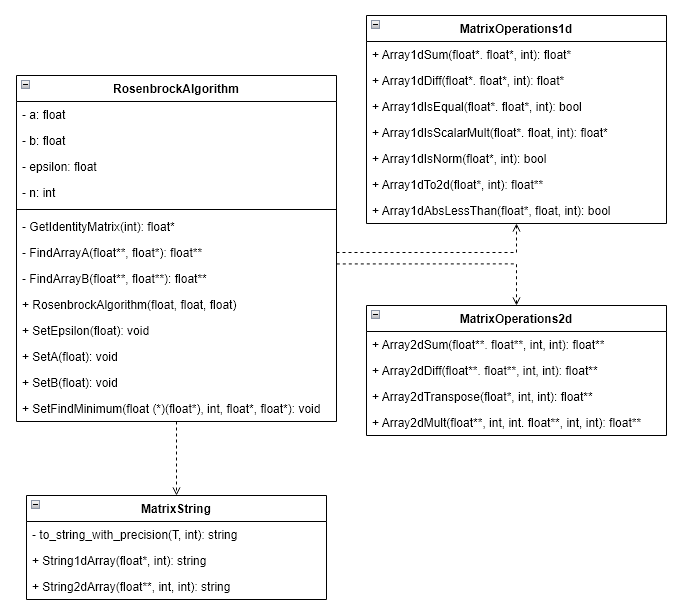


При применении объектно-ориентированного подхода алгоритм оптимизации был выделен в отдельный класс, содержащий защищенные поля и методы.

Поля учувствуют в работе алгоритма и сильно влияют на результаты алгоритма, поэтому для их доступа было принято решение создать отдельные методы, позволяющие изменять параметры алгоритма, например, с целью подбора их оптимальных значений, а также тестирования алгоритма при различных значениях коэффициентов сжатия или растяжения.

Метод определения минимальной точки содержит основной блок, выполняющий шаги алгоритма. При этом пятый шаг алгоритма, содержащий сложные операции с матрицами, был упрощен введением защищенных методов поиска компонентов А и B, необходимых для вычисления новых направлений поиска минимума.

*Рисунок 7 - UML диаграмма для программы, написанной с применением объектно-ориентированного подхода.*



**Вывод**

В ходе практической работы был изучен алгоритм оптимизации методом Розенброка с дискретным шагом. Была дана формулировка алгоритма и рассчитаны итерации алгоритма, а также составлен график этих итераций для функции двух переменных F1.

Для написания кода программы процедурным подходом была составлена блок-схема и псевдокод. При помощи программы был рассчитан минимум целевых функций F1 и F2.

Для функции F1 минимум рассчитан в точке (5;1) со значением функции = -481. Данное значение получено в результате 7 итераций алгоритма Розенброка. Для функции F2 минимум лежит в точке (0.5; -1; 0) со значением функции = -1.25. На нахождение этого минимума потребовалось 10 итераций.

При создании программы с применением модульного подхода алгоритм был разбит на 1 основной и 3 независимых модуля, содержащих операции для работы с матрицами. Результаты выполнения модульной программы не отличаются от процедурной, при этом использование модулей позволило значительно упростить читаемость кода алгоритма оптимизации.

Была написана программа с применением объектно-ориентированного подхода. Алгоритм программы был помещен в класс, содержащий скрытые методы, участвующие вычислении минимума функции. Результаты выполнения объектно-ориентированной программы не отличаются от процедурной или модульной, однако она обладает наивысшей читаемостью и расширяемостью среди трех.

**Приложение**

Исходный код процедурной программы:

#include <iostream>

#include <math.h>

using namespace std;

#define MAX\_ITERATIONS 20

float\* Rosenbrock(int n, float (\*function)(float\*), float epsilon, float a, float b, float\* delta, float\* x1)

{

float\*\* s = new float\*[n];

for (int i = 0; i < n; ++i)

s[i] = new float[n]();

for (int j = 0; j < n; j++)

for (int i = 0; i < n; i++)

if (i == j)

s[i][j] = 1.0f;

float\* y = x1;

int k = 1;

int j = 0;

float \*y\_with\_step;

float\* y\_successful = y;

cout << ">>> Iteration: 1" << endl;

while (k < MAX\_ITERATIONS)

{

for (j = 0; j < n; j++)

{

// Шаг 1

cout << ">>> Step 1" << endl;

// Вывод данных об итерации

cout << "y = [ ";

for (int i = 0; i < n; i++){

cout << y[i];

if (i != n-1)

cout << ", ";

}

cout << " ]" << endl;

float f\_y = function(y);

cout << "> f(y) = " << f\_y << endl;

cout << "> delta[" << j <<"] =" << delta[j] << endl;

// Вычисление шага

float \*step = new float[n]();

for (int i = 0; i < n; i++)

step[i] = s[j][i] \* delta[j];

//Прибавление шага к текцщей координате

y\_with\_step = new float[n]();

for (int i = 0; i < n; i++)

y\_with\_step[i] = y[i] + step[i];

//Вывод шага и значения функции в нем

cout << "y\_with\_step = [ ";

for (int i = 0; i < n; i++){

cout << y\_with\_step[i];

if (i != n-1)

cout << ", ";

}

cout << " ]" << endl;

float f\_y\_with\_step = function(y\_with\_step);

cout << "> f(y\_with\_step) = " << f\_y\_with\_step << endl;

if (f\_y\_with\_step < f\_y)

{

cout << "Step successful." << endl;

y = y\_with\_step;

y\_successful = y\_with\_step;

delta[j] \*= a;

}

else

{

cout << "Step unsuccessful." << endl;

delta[j] \*= b;

}

cout << "> delta[" << j << "] = " << delta[j] << endl;

// Шаг 2

cout << ">>> Step 2" << endl;

cout << "> j = " << j << endl;

}

j = 0;

// Шаг 3

cout << ">>> Step 3" << endl;

bool y\_equalto\_y\_with\_step = true;

for (int i = 0; i < n; i++)

if (y[i] != y\_with\_step[i])

y\_equalto\_y\_with\_step = false;

if (!y\_equalto\_y\_with\_step)

{

y = y\_successful;

continue;

}

// Шаг 4

cout << ">>> Step 4" << endl;

bool abs\_delta\_lessthan\_epsilon = true;

for (int i = 0; i < n; i++)

if (abs(delta[i]) >= epsilon)

abs\_delta\_lessthan\_epsilon = false;

if (abs\_delta\_lessthan\_epsilon)

{

cout << "> Epsilon check successful. Finishing iterations." << endl;

break;

}

// Шаг 5

cout << ">>> Step 5" << endl;

// Вычисление новой делты

delta = new float[n]();

for (int i = 0; i < n; i++)

delta[i] = y\_successful[i] - x1[j];

cout << "> New delta = [ ";

for (int i = 0; i < n; i++){

cout << delta[i];

if (i != n-1)

cout << ", ";

}

cout << " ]" << endl;

// Вычисление А

float\*\* a\_array = new float\*[n];

for (int j = 0; j < n; j++)

{

float\* aj = new float[n]();

for (int i = j; i < n; i++)

{

float\* addition = new float[n]();

for (int m = 0; m < n; m++)

addition[m] = (s[i])[m] \* delta[i];

for (int m = 0; m < n; m++)

aj[m] = aj[m] + addition[m];

}

a\_array[j] = aj;

}

// Вычисление В

float\*\* b\_array = new float\*[n];

b\_array[0] = a\_array[0];

float a\_array0\_norm;

float norm\_sum = 0.;

for (int i = 0; i < n; i++)

{

norm\_sum += pow(a\_array[0][i], 2.0);

}

a\_array0\_norm = sqrt(norm\_sum);

s[0] = new float[n];

for (int i = 0; i < n; i++)

s[0][i] = a\_array[0][i] / a\_array0\_norm;

for (int j = 1; j < n; j++)

{

// Перевод одномерного массива a\_array[j] в двумерный

float \*\*A = new float\*[n]();

for (int m = 0; m < n; m++)

A[m] = new float[1]();

for (int m = 0; m < n; m++)

{

A[0][m] = a\_array[j][m];

}

//Вычисжегие B[j]

float \*\*b\_sum = new float\*[1];

b\_sum[0] = new float[n]();

for (int i = 0; i < j; i++)

{

// Перевод одномерного массива s[i] в двумерный

float\*\* s\_i = new float\*[n]();

for (int m = 0; m < n; m++)

s\_i[m] = new float[1]();

for (int m = 0; m < n; m++)

{

s\_i[0][m] = s[i][m];

}

//Транспонирование s\_i

float \*\*s\_i\_transposed = new float\*[1];

int o;

for (int m = 0; m < n; m++)

s\_i\_transposed[m] = new float[1]();

for (int m = 0; m < n; m++)

for (o = 0; o < n; o++)

s\_i\_transposed[m][o] = s\_i[o][m];

//Умножение A на транспонированую s\_i

float \*\*a\_s\_transposed\_mul = new float\*[1]{new float[1]()};

for(int m=0; m<n; ++m)

a\_s\_transposed\_mul[0][0] += A[0][m] \* s\_i\_transposed[m][0];

// Умножаем результат предыдущего умножения на s\_i

float \*\*prod\_si\_mul = new float\*[1]{new float[1]()};

for(int m=0; m<n; ++m)

prod\_si\_mul[0][m] = s\_i[0][m] \* a\_s\_transposed\_mul[0][0];

// Прибавляем результат предыдущего умножения к b\_sum

for(int m=0; m<n; ++m)

b\_sum[0][m] += prod\_si\_mul[0][m];

}

float \*B = new float[n]();

for(int m=0; m<n; ++m)

B[m] = A[0][m] - b\_sum[0][m];

b\_array[j] = B;

float b\_norm = 0;

float b\_norm\_sum = 0;

for (int i = 0; i < n; i++)

{

b\_norm\_sum += pow(B[i], 2.0);

}

b\_norm = sqrt(b\_norm\_sum);

// Установка новго направления

for(int m=0; m<n; ++m)

s[j][m] = B[m] / b\_norm;

}

k += 1;

cout << ">>> Iteration: " << k << endl;

}

if (k == MAX\_ITERATIONS)

{

cout << "Warning: Max iterations exceeded!" << endl;

}

cout << "> Finished" << endl;

cout << "> Iterations:" << k << endl;

cout << "> Result: [ ";

for (int i = 0; i < n; i++){

cout << y[i];

if (i != n-1)

cout << ", ";

}

cout << " ]" << endl;

cout << "> Function:" << function(y) << endl;

return y;

}

float function1(float\* x)

{

return 9 \* pow(x[0], 2) + 16 \* pow(x[1], 2) - 90 \* x[0] - 128 \* x[1];

}

float function2(float\* x)

{

return pow(x[0], 2) + 2 \* x[0] \* x[1] + 2 \* pow(x[1], 2) + pow(x[2], 2) - x[1] \* x[2] + x[0] + 3 \* x[1] - x[2];

}

int main() {

float epsilon = 0.01;

float a = 3.0;

float b = -0.5;

int n1 = 2;

int n2 = 3;

float\* x1 = new float[n1] {2.5f, 2.5f};

float\* x2 = new float[n2] {2.5f, 2.5f, 2.5f};

float\* delta1 = new float[n1] {0.5f, 0.5f};

float\* delta2 = new float[n2] {0.5f, 0.5f, 0.5f};

Rosenbrock(n1, function1, epsilon, a, b, delta1, x1);

return 0;

}