Министерство науки и высшего образования Российской Федерации федеральное государственное автономное образовательное учреждение высшего образования **«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

Отчет

по **Лабораторной работе №4**

по дисциплине **«Методы Оптимизации»**

Работа учеников

группы М32351  
Юльцовой Натальи,  
Колчина Дмитрия

Вариант 1

2021

**Цель и задачи работы:**

Цель: Разработать программы для безусловной минимизации функций многих переменных.

Задачи:

Реализовать алгоритмы

1. Метод Ньютона: а) классический, б) с одномерным поиском, в) с направлением спуска.

1.1. Продемонстрируйте работу методов на 2-3 функциях, в том числе на не квадратичных.

* Для поиска ньютоновского направления спуска необходимо использовать прямой или итерационный метод решения СЛАУ (даже, если она размерности 2).
* Результаты иллюстрируйте траекториями спуска.
* Укажите количество итераций, необходимых для достижения заданной точности.
* В случае одномерного поиска указывайте найденные значения параметра.
* Проведите исследование влияние выбора начального приближения на результат (не менее трех).

1.2. Исследуйте работу методов на двух функциях с заданным начальным приближением

𝑓(𝑥) = 𝑥12 + 𝑥22 − 1.2𝑥1𝑥2, 𝑥0 = (4, 1)𝑇;

𝑓(𝑥) = 100(𝑥2 − 𝑥12)2 + (1 − 𝑥1)2, 𝑥0 = (−1.2, 1)𝑇.

* Для поиска ньютоновского направления спуска необходимо использовать прямой или итерационный метод решения СЛАУ (даже, если она размерности 2).
* Сравните результаты с минимизацией методом наискорейшего спуска (из лаб. работы 2).
* Постройте таблицу или график зависимости «метод : количество итераций».
* Для каждого метода приведите иллюстрации траекторий сходимости.

2. Квазиньютоновский метод

метод Давидона-Флетчера-Пауэлла и метод Пауэлла(вариант 1).

Работу квазиньютоновских методов сравните с наилучшим методом Ньютона (по результатам 1.2) на функциях:

f(x) = 100(x2 - x12)2 + (1 - x1)2

f(x) = (x2 + x12 - 11)2 + (x1 + x22 - 7)2

* Для каждого метода приведите иллюстрации траекторий сходимости.
* Проведите исследование влияние выбора начального приближения на результат (не менее трех), оцените скорость сходимости.
* Постройте таблицу или график зависимости «метод : количество итераций».

Для отображения траектории поиска точки используйте ранее реализованную графическую систему или сторонние программы.

Ход работы.

Задание 1. Метод Ньютона

В работе был использован в качестве одномерного поиска метод дихотомии.

1.1

1) f(x) = (x + 1)2 + (y - 1)2

Классический метод

Начальные данные (10, 15).

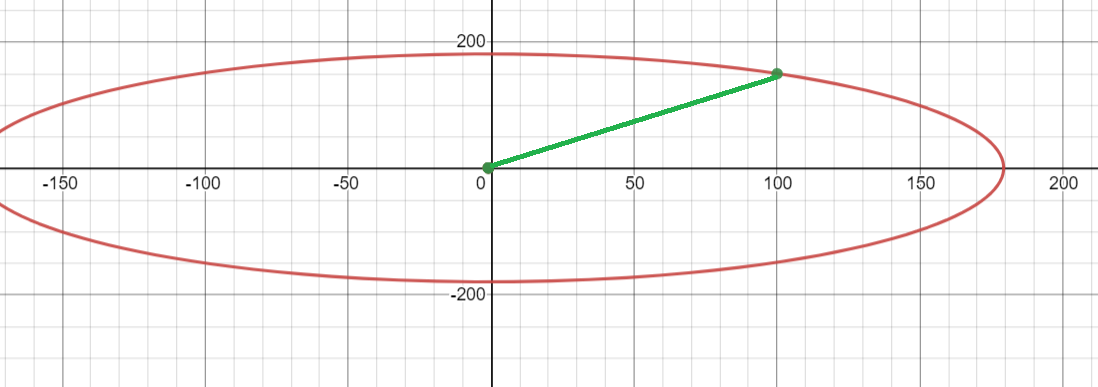
|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | 10 | 15 |
| 1 | -1 | 1 |
| 2 | -1 | 1 |

Изображение выглядит как спорт, ракетбол, спортивная игра

Автоматически созданное описание

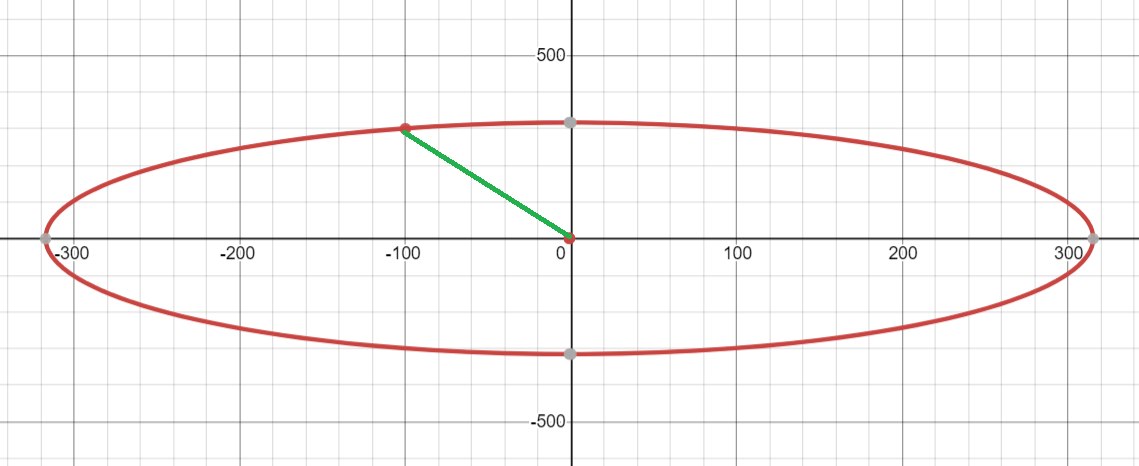
Начальные данные (100, 150).

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | 100 | 150 |
| 1 | -1 | 1 |
| 2 | -1 | 1 |



Начальные данные (-100, 300).

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | -100 | 300 |
| 1 | -1 | 1 |
| 2 | -1 | 1 |



Метод с одномерным поиском

Начальные данные (10, 15).

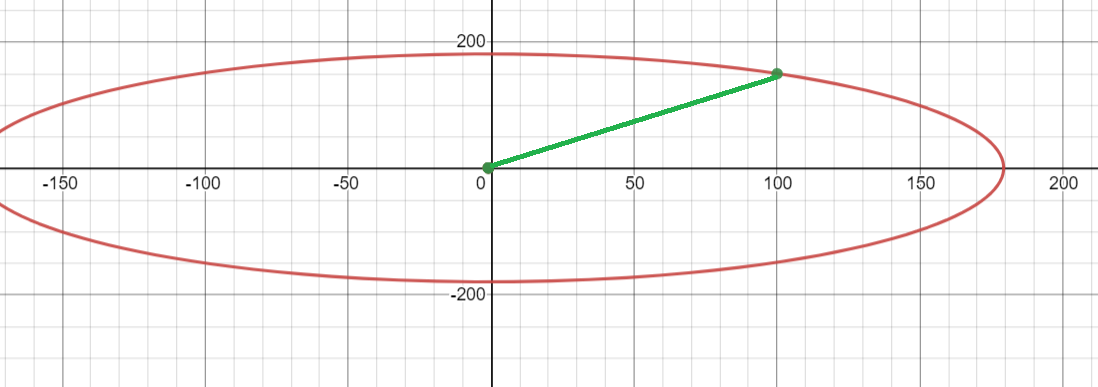
|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 10.0000000000 | 15.0000000000 |  |
| 1 | -0.9999999459 | 1.0000000688 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |
| 3 | -1.0000000000 | 1.0000000000 | -93.7499999918 |

Изображение выглядит как спорт, ракетбол, спортивная игра

Автоматически созданное описание

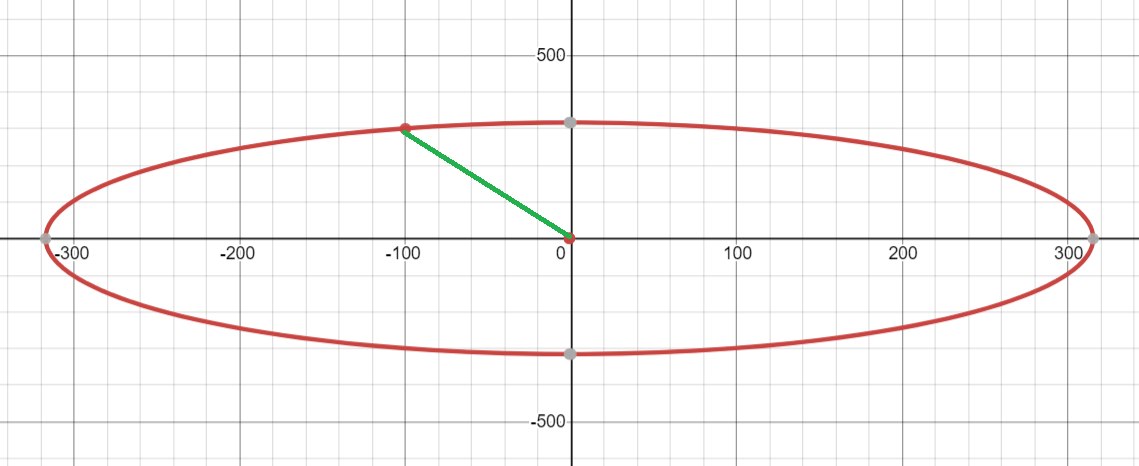
Начальные данные (100, 150).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 100.0000000000 | 150.0000000000 |  |
| 1 | -0.9999995036 | 1.0000007323 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |
| 3 | -1.0000000000 | 1.0000000000 | -92.9687499919 |



Начальные данные (-100, 300).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -100.0000000000 | 300.0000000000 |  |
| 1 | -1.0000004865 | 1.0000014694 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |
| 3 | -1.0000000000 | 1.0000000000 | -87.4999999920 |



Метод с направлением спуска

Начальные данные (10, 15).

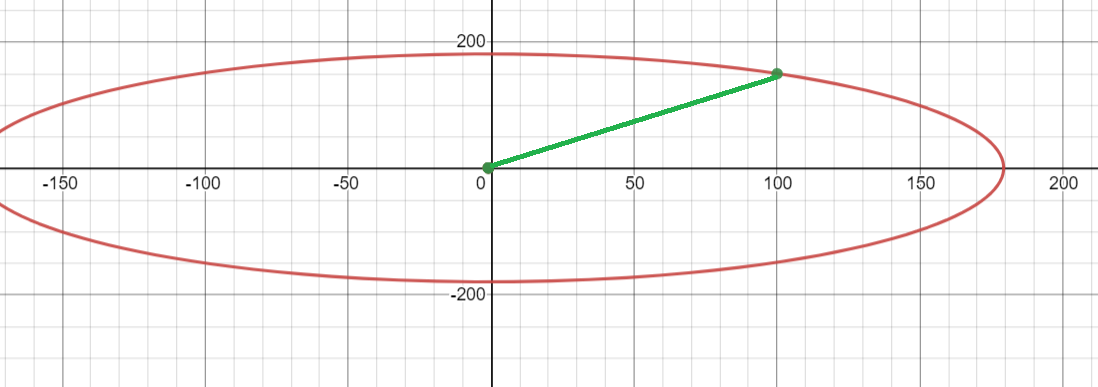
|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 10.0000000000 | 15.0000000000 |  |
| 1 | -0.9999999459 | 1.0000000688 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |

Изображение выглядит как спорт, ракетбол, спортивная игра

Автоматически созданное описание

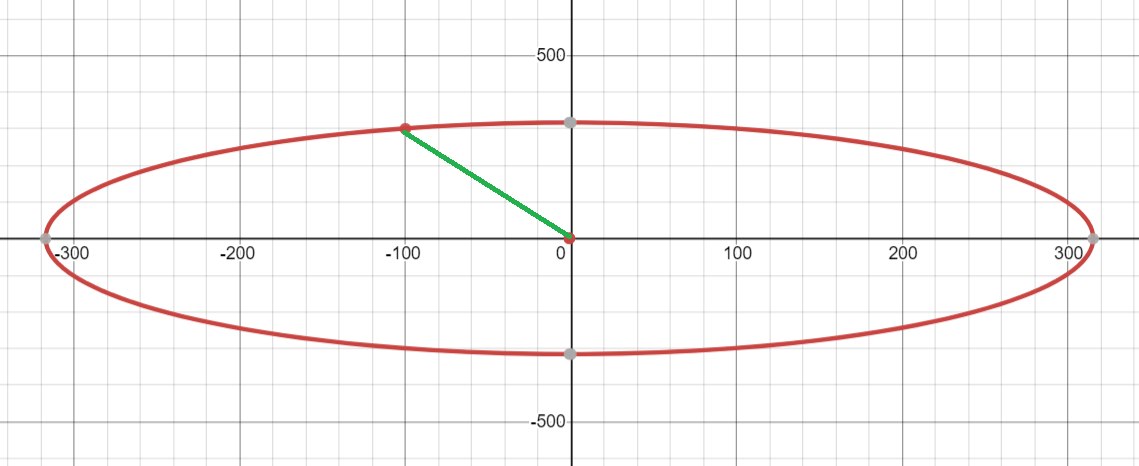
Начальные данные (100, 150).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 100.0000000000 | 150.0000000000 |  |
| 1 | -0.9999995036 | 1.0000007323 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |



Начальные данные (-100, 300).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -100.0000000000 | 300.0000000000 |  |
| 1 | -1.0000004865 | 1.0000014694 | 0.9999999951 |
| 2 | -1.0000000000 | 1.0000000000 | 0.9999999951 |



1. f(x) = (x2 + y2 + 2 – у)2

Классический метод

Начальные данные (10, 15).

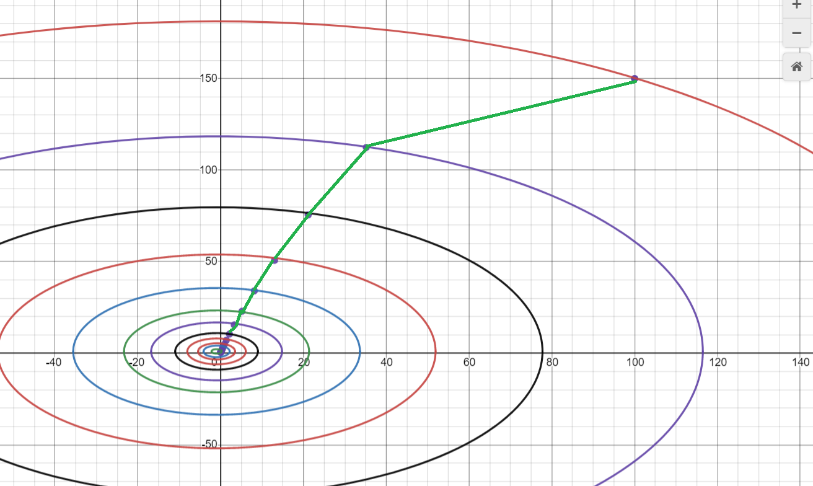
|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | 10.0000000000 | 15.0000000000 |
| 1 | 3.8811993211 | 11.2463511220 |
| 2 | 2.3345116075 | 7.6878075067 |
| 3 | 1.4531922282 | 5.2615123115 |
| 4 | 0.9215923827 | 3.6061441891 |
| 5 | 0.5858106946 | 2.4610679408 |
| 6 | 0.3603431913 | 1.6446168420 |
| 7 | 0.1939824418 | 1.0398124179 |
| 8 | 0.0654400413 | 0.6283032421 |
| 9 | 0.0056113234 | 0.5028990091 |
| 10 | 0.0000358085 | 0.5000001315 |
| 11 | 0.0000000015 | 0.5000000000 |
| 12 | 0.0000000000 | 0.5000000000 |

Изображение выглядит как спортивная игра, спорт

Автоматически созданное описание

Начальные данные (100, 150).

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | 100.0000000000 | 150.0000000000 |
| 1 | 35.1598332448 | 112.3905387363 |
| 2 | 21.1371648398 | 75.5573312981 |
| 3 | 13.0080281325 | 50.7313179915 |
| 4 | 8.1279735193 | 34.0718247906 |
| 5 | 5.1366215957 | 22.9144860207 |
| 6 | 3.2766042571 | 15.4481268803 |
| 7 | 2.1072884269 | 10.4509405886 |
| 8 | 1.3649943766 | 7.1011907022 |
| 9 | 0.8886756498 | 4.8463264282 |
| 10 | 0.5781876427 | 3.3137742732 |
| 11 | 0.3699096743 | 2.2507485063 |
| 12 | 0.2221749393 | 1.4857128572 |
| 13 | 0.1081874074 | 0.9169109055 |
| 14 | 0.0255576357 | 0.5667431106 |
| 15 | 0.0008486065 | 0.5003848775 |
| 16 | 0.0000000000 | 0.5000000000 |
| 17 | 0.0000000000 | 0.5000000000 |
| 18 | 100.0000000000 | 150.0000000000 |



Начальные данные (-100, 300).

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | -100.0000000000 | 300.0000000000 |
| 1 | -58.8951441364 | 201.8333116552 |
| 2 | -35.7467220854 | 135.3871187927 |
| 3 | -22.0655173802 | 90.7267259722 |
| 4 | -13.7673506043 | 60.7976528515 |
| 5 | -8.6512171072 | 40.7709343556 |
| 6 | -5.4610435208 | 27.3809888337 |
| 7 | -3.4551349454 | 18.4311709551 |
| 8 | -2.1855829515 | 12.4476554242 |
| 9 | -1.3775623196 | 8.4425917554 |
| 10 | -0.8604208717 | 5.7535988703 |
| 11 | -0.5271970046 | 3.9354572931 |
| 12 | -0.3104028234 | 2.6872117592 |
| 13 | -0.1675156415 | 1.8039112479 |
| 14 | -0.0729461085 | 1.1505804178 |
| 15 | -0.0165593948 | 0.6838211007 |
| 16 | -0.0003024045 | 0.5067624600 |
| 17 | 0.0000000887 | 0.5000003541 |
| 18 | 0.0000000000 | 0.5000000000 |
| 19 | 0.0000000000 | 0.5000000000 |

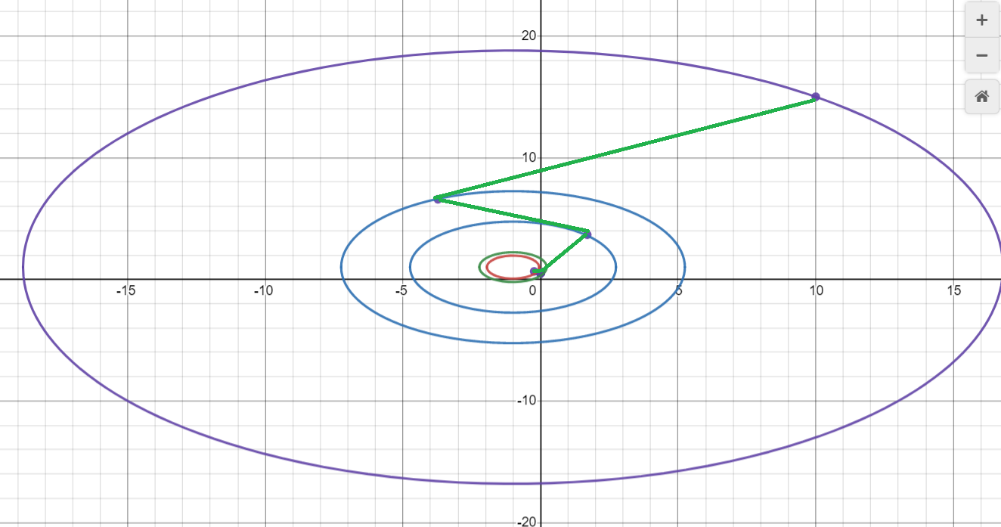
Изображение выглядит как спорт, спортивная игра, ракетбол

Автоматически созданное описание

Метод с одномерным поиском

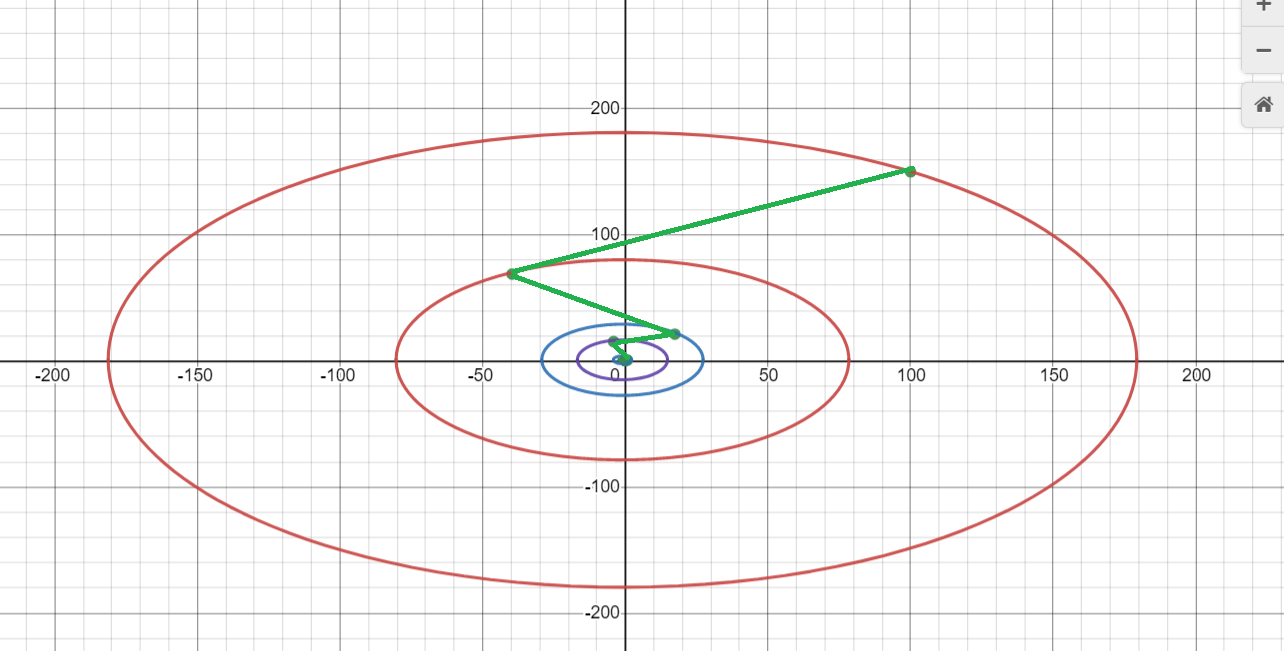
Начальные данные (10, 15).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 10.0000000000 | 15.0000000000 |  |
| 1 | -3.7286282882 | 6.5780145694 | 2.2436796047 |
| 2 | 1.6988295919 | 3.6624523303 | 1.9230914942 |
| 3 | -0.2331007494 | 0.6494644141 | 2.7031429752 |
| 4 | -0.0001931994 | 0.5000778427 | 13.2568573518 |
| 5 | -0.0001931994 | 0.5000778427 | -0.0000000058 |



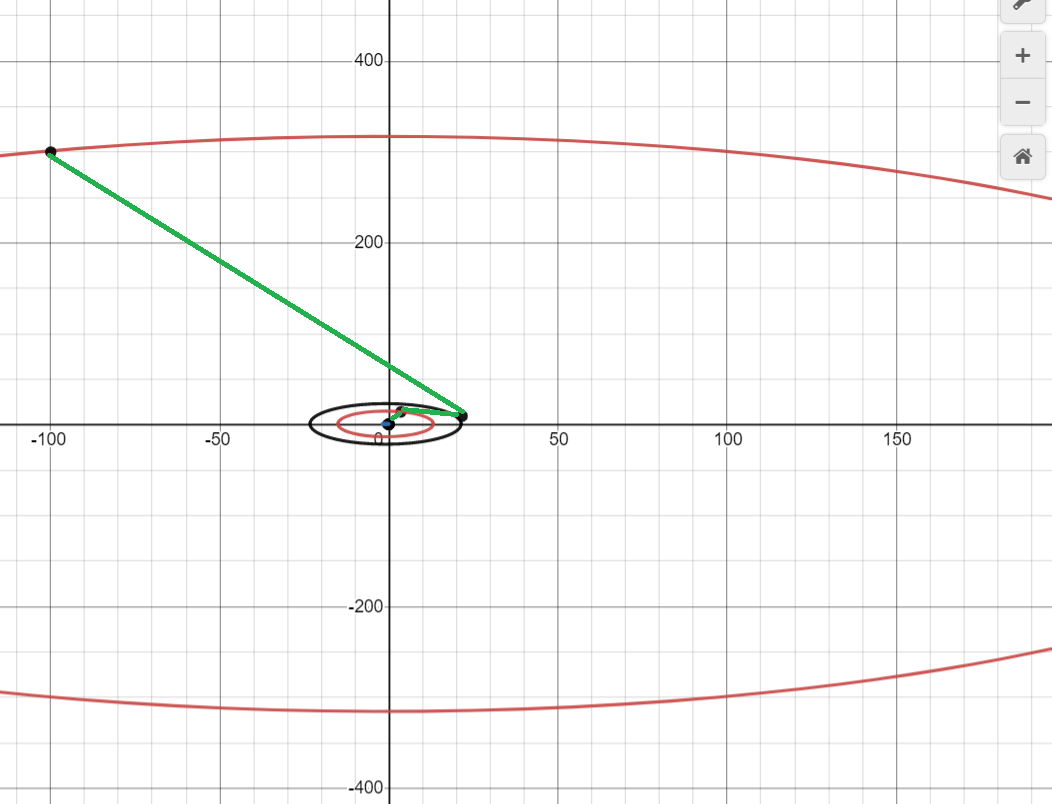
Начальные данные (100, 150).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 100.0000000000 | 150.0000000000 |  |
| 1 | -39.7108505427 | 68.9630427474 | 2.1546960400 |
| 2 | 17.3785710448 | 21.3276233959 | 2.4603640136 |
| 3 | -4.0899603567 | 15.4571617689 | 1.6251095046 |
| 4 | 0.7655850365 | 0.7528048173 | 2.9632790421 |
| 5 | -0.0000037413 | 0.4999980582 | 0.9924590646 |
| 6 | -0.0000037413 | 0.4999980582 | 0.0000122993 |



Начальные данные (-100, 300).

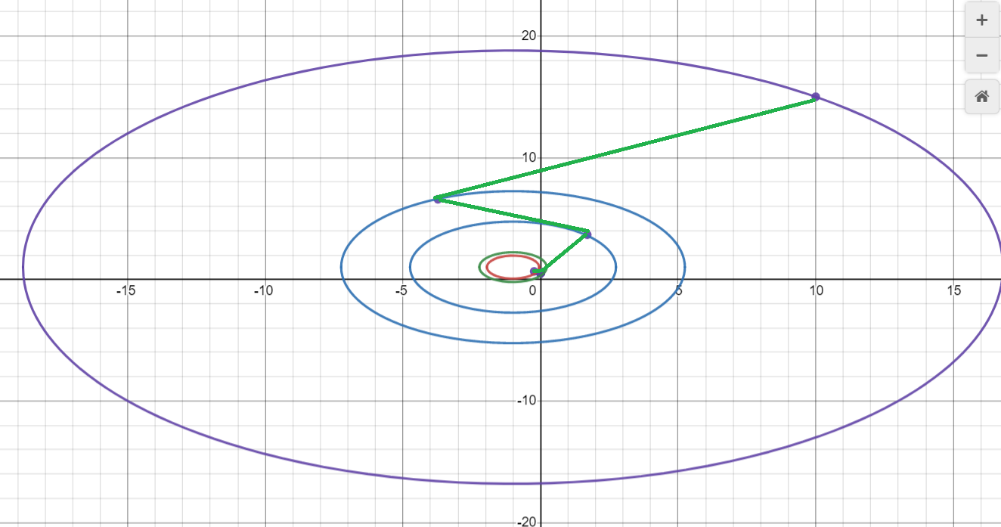
|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -100.0000000000 | 300.0000000000 |  |
| 1 | 21.6178867790 | 9.5519510348 | 2.9587231052 |
| 2 | 3.5505382848 | 14.2256091761 | 0.7477829756 |
| 3 | -0.3222785217 | 0.5915446623 | 2.9749604000 |
| 4 | 0.0063046741 | 0.5409255405 | 0.6716499629 |
| 5 | 0.0000437356 | 0.4999931951 | 1.0021239577 |
| 6 | 0.0002123158 | 0.5000005538 | 6.3038841994 |
| 7 | 0.0002123158 | 0.5000005538 | -0.0000000058 |



Метод с направлением спуска

Начальные данные (10, 15).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 10.0000000000 | 15.0000000000 |  |
| 1 | -3.7286282882 | 6.5780145694 | 2.2436796047 |
| 2 | 1.6988295919 | 3.6624523303 | 1.9230914942 |
| 3 | -0.2331007494 | 0.6494644141 | 2.7031429752 |
| 4 | -0.0001931994 | 0.5000778427 | 13.2568573518 |
| 5 | -0.0001931994 | 0.5000778427 | -0.0000000058 |



Начальные данные (100, 150).

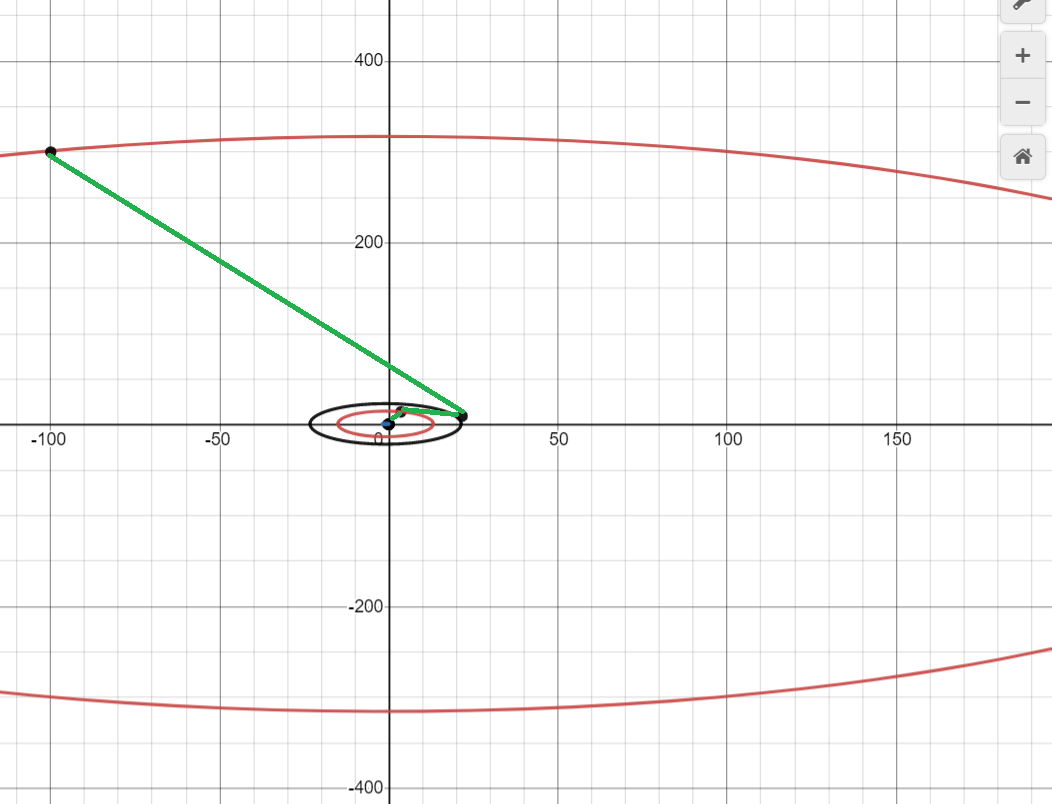
|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 100.0000000000 | 150.0000000000 |  |
| 1 | -39.7108505427 | 68.9630427474 | 2.1546960400 |
| 2 | 17.3785710448 | 21.3276233959 | 2.4603640136 |
| 3 | -4.0899603567 | 15.4571617689 | 1.6251095046 |
| 4 | 0.7655850365 | 0.7528048173 | 2.9632790421 |
| 5 | -0.0000037413 | 0.4999980582 | 0.9924590646 |
| 6 | -0.0000037413 | 0.4999980582 | 0.0000122993 |

Изображение выглядит как ракетбол, спорт

Автоматически созданное описание

Начальные данные (-100, 300).

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -100.0000000000 | 300.0000000000 |  |
| 1 | 21.6178867790 | 9.5519510348 | 2.9587231052 |
| 2 | 3.5505382848 | 14.2256091761 | 0.7477829756 |
| 3 | -0.3222785217 | 0.5915446623 | 2.9749604000 |
| 4 | 0.0063046741 | 0.5409255405 | 0.6716499629 |
| 5 | 0.0000437356 | 0.4999931951 | 1.0021239577 |
| 6 | 0.0002123158 | 0.5000005538 | 6.3038841994 |
| 7 | 0.0002123158 | 0.5000005538 | -0.0000000058 |

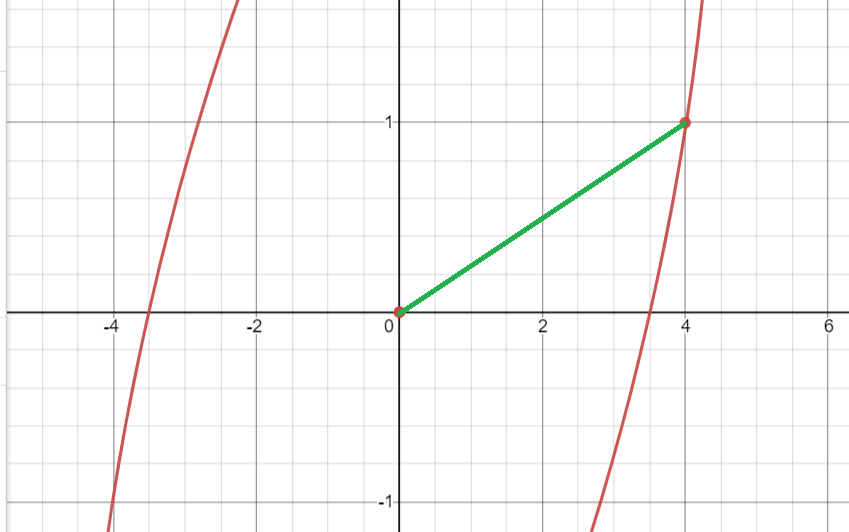


1.2

1) 𝑓(𝑥) = 𝑥12 + 𝑥22 − 1.2𝑥1𝑥2, 𝑥0 = (4, 1)𝑇;

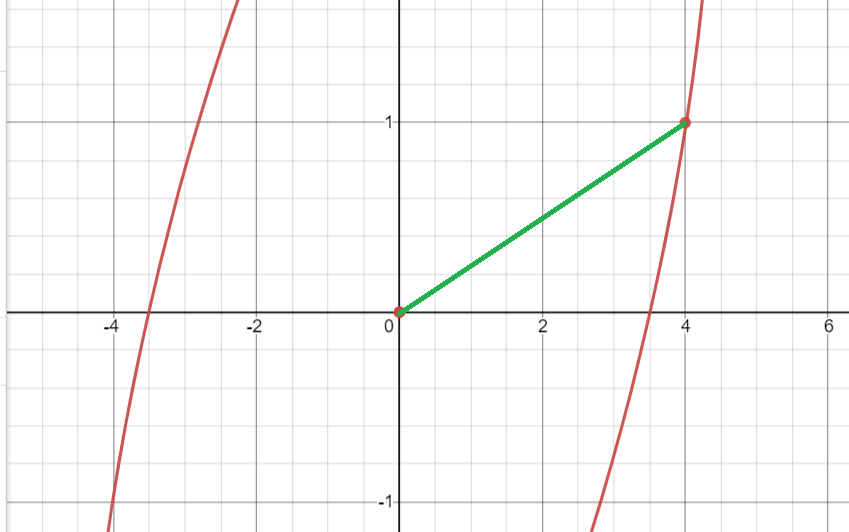
Классический

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | 4.0000000000 | 1.0000000000 |
| 1 | 0.0000000000 | -0.0000000000 |
| 2 | 0.0000000000 | 0.0000000000 |



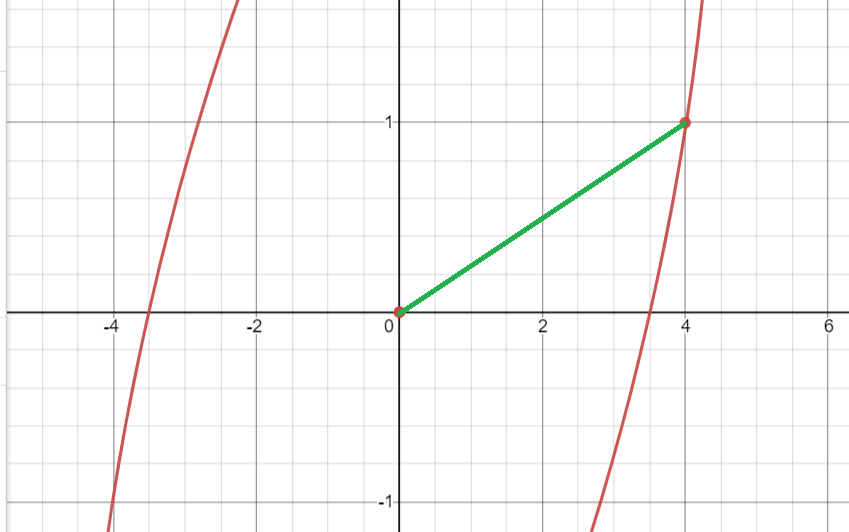
С одномерным поиском

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 4.0000000000 | 1.0000000000 |  |
| 1 | -0.0000199272 | -0.0000049818 | 1.0000049818 |
| 2 | -0.0000056495 | -0.0000014124 | -1.0943296847 |



С направлением спуска

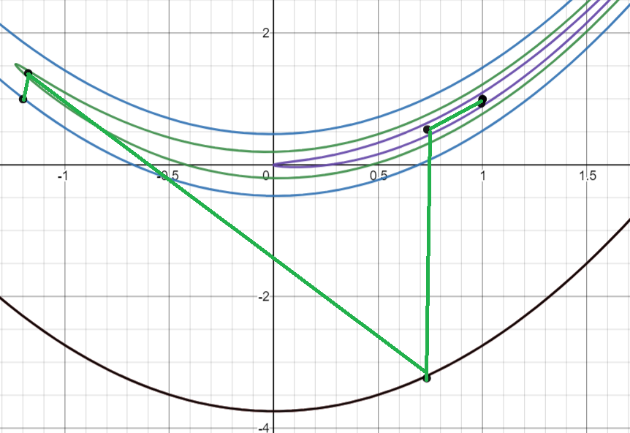
|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | 4.0000000000 | 1.0000000000 |  |
| 1 | -0.0000199272 | -0.0000049818 | 1.0000049818 |
| 2 | -0.0000007180 | -0.0000001795 | 0.9639680145 |



1. 𝑓(𝑥) = 100(𝑥2 − 𝑥12)2 + (1 − 𝑥1)2, 𝑥0 = (−1.2, 1)𝑇.

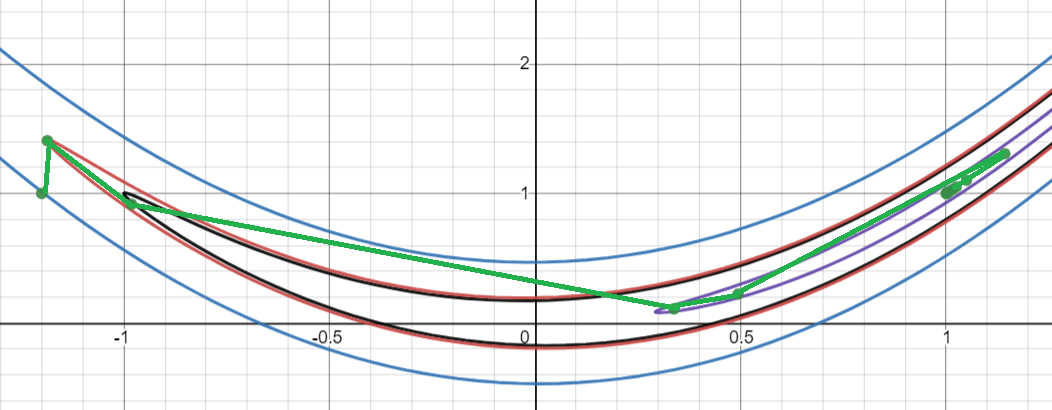
Классический

|  |  |  |
| --- | --- | --- |
| Итерация | x | y |
| 0 | -1.2 | 1 |
| 1 | -1.1753939388 | 1.3933394089 |
| 2 | 0.7328322332 | -3.239238384 |
| 3 | 0.7354875844 | 0.5384849292 |
| 4 | 0.9934943999 | 0.9348489284 |
| 5 | 0.9999999349 | 0.9999996499 |
| 6 | 0.9999999999 | 0.9999999998 |



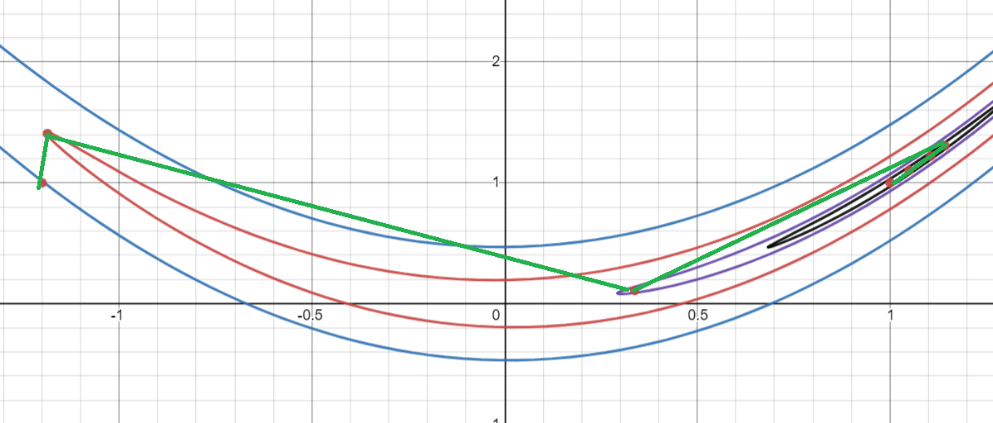
С одномерным поиском

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -1.2000000000 | 1.0000000000 |  |
| 1 | -1.1863084580 | 1.4080079507 | 1.0019719350 |
| 2 | -0.9813701314 | 0.9172262086 | 6.6769300222 |
| 3 | 0.3377983197 | 0.1089675355 | 38.8330271073 |
| 4 | 0.4937493397 | 0.2236890522 | 1.8212496779 |
| 5 | 1.1424391953 | 1.3074867276 | 22.0513159484 |
| 6 | 1.0495451157 | 1.1025532222 | 3.1994282399 |
| 7 | 1.0244270759 | 1.0499836591 | 3.3882080662 |
| 8 | 1.0139319553 | 1.0283713306 | 3.4775673474 |
| 9 | 1.0052870357 | 1.0107240473 | 3.5566150253 |
| 10 | 1.0023357443 | 1.0046573646 | 1.3504564425 |
| 11 | 1.0001926493 | 1.0003837251 | 1.3471424243 |
| 12 | 1.0001056687 | 1.0002104654 | 1.3469278476 |



С направлением спуска

|  |  |  |  |
| --- | --- | --- | --- |
| Итерация | x | y | Параметр |
| 0 | -1.2000000000 | 1.0000000000 |  |
| 1 | -1.1863084580 | 1.4080079507 | 1.0019719350 |
| 2 | 0.3377983197 | 0.1089675355 | 38.8330271073 |
| 3 | 1.1424391953 | 1.3074867276 | 22.0513159484 |
| 4 | 1.1057069105 | 1.2216234407 | 1.5026151758 |
| 5 | 1.0495451157 | 1.1025532222 | 3.1994282399 |
| 6 | 1.0002600959 | 1.0005180851 | 1.3472139499 |



Метод наискорейшего спуска работает в разы дольше.

Сравнение методов

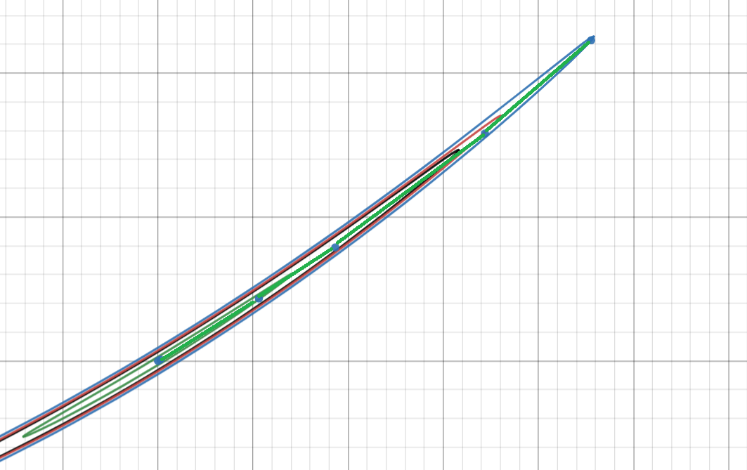
|  |  |  |
| --- | --- | --- |
| Функция  Метод | 1 | 2 |
| Классический | 2 | 6 |
| С одномерным поиском | 2 | 12 |
| С направлением спуска | 2 | 6 |

Задание 2. Квазиньютоновский метод

Метод Давидона-Флэтчер-Пауэлла

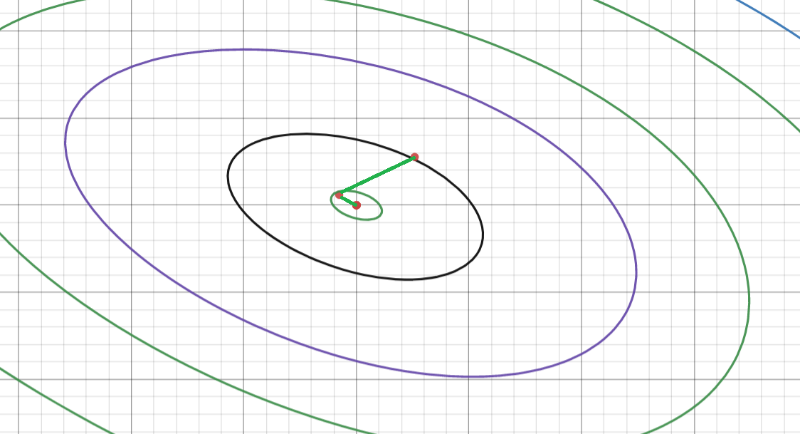
Начальные данные 2 2

f(x) = 100(x2 - x12)2 + (1 - x1)2



11 итераций

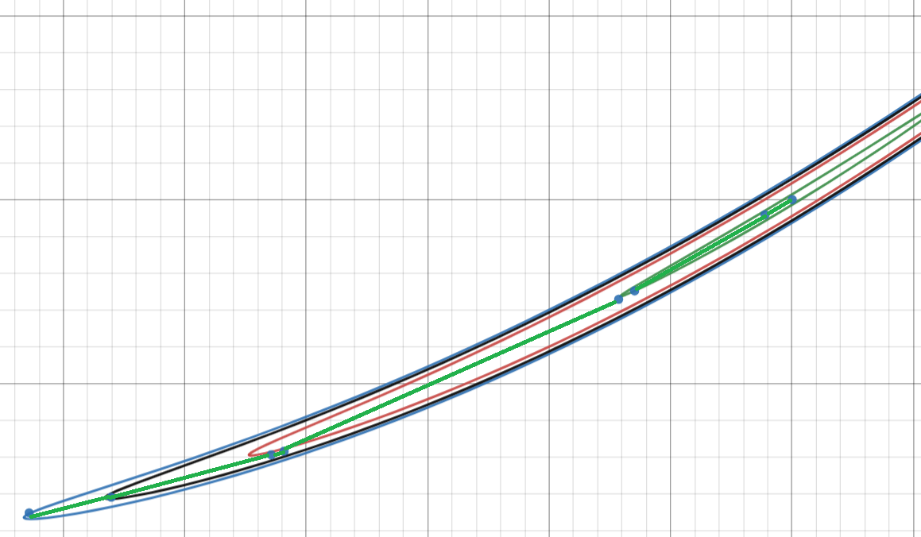
f(x) = (x2 + x12 - 11)2 + (x1 + x22 - 7)2



9 итераций

Начальные данные -0.5 3

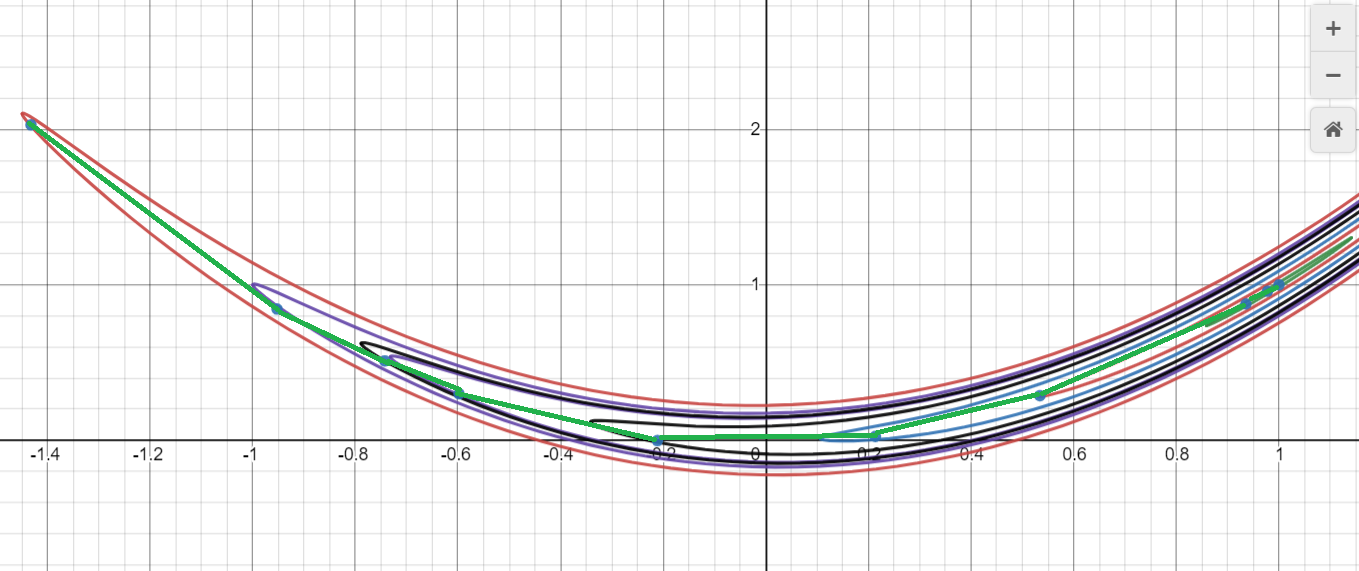
f(x) = 100(x2 - x12)2 + (1 - x1)2



16 итераций

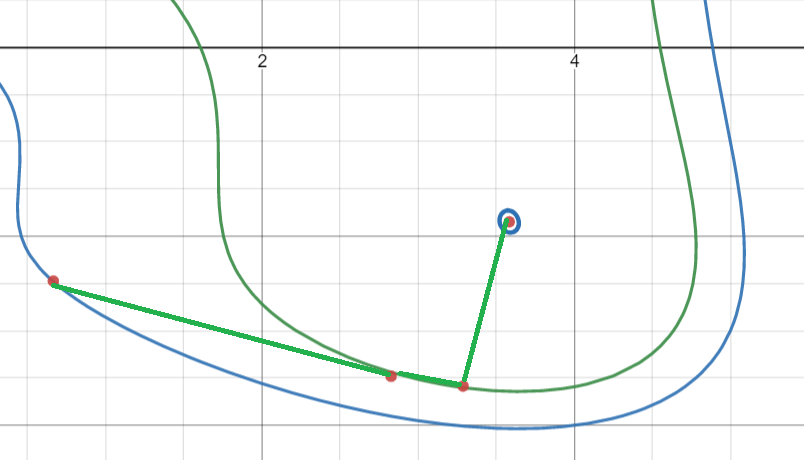
Начальные данные -2 2

f(x) = 100(x2 - x12)2 + (1 - x1)2



11 итераций

f(x) = (x2 + x12 - 11)2 + (x1 + x22 - 7)2

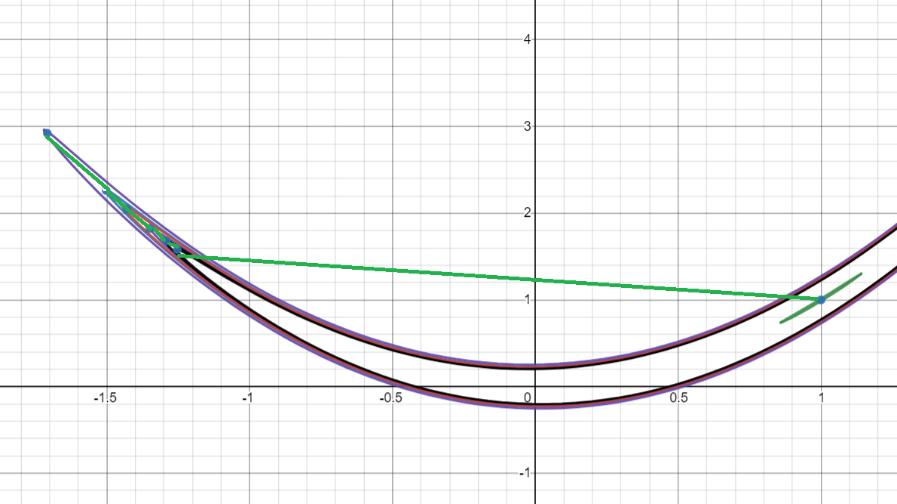


5 итераций

Метод Пауэлла

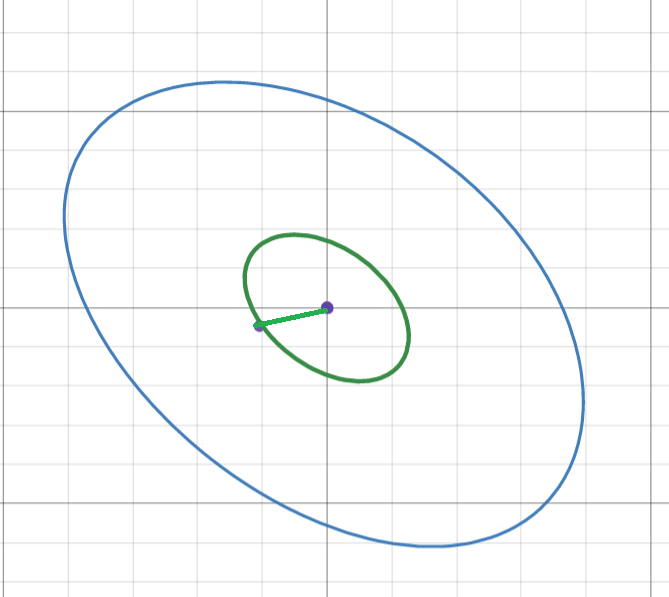
Начальные данные 2 2

f(x) = 100(x2 - x12)2 + (1 - x1)2



11 итераций

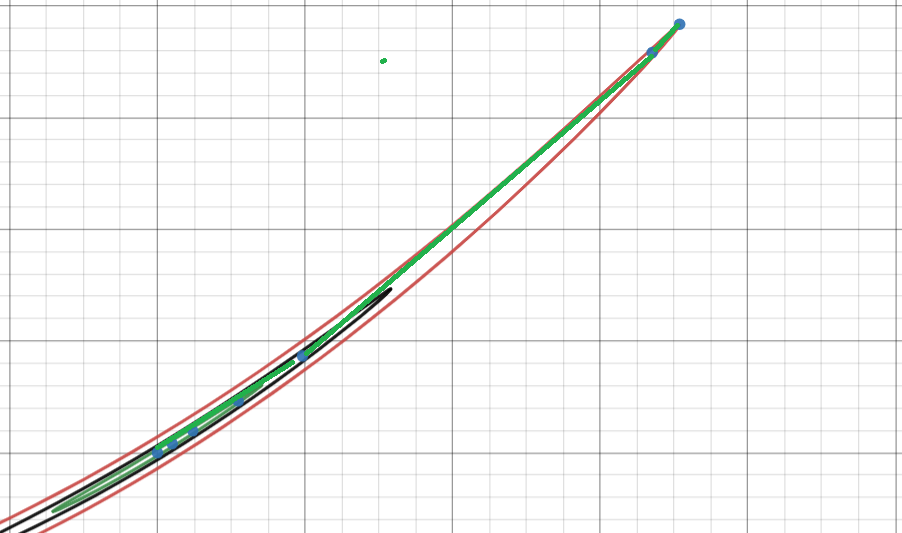
f(x) = (x2 + x12 - 11)2 + (x1 + x22 - 7)2



9 итераций

Начальные данные -0.5 3

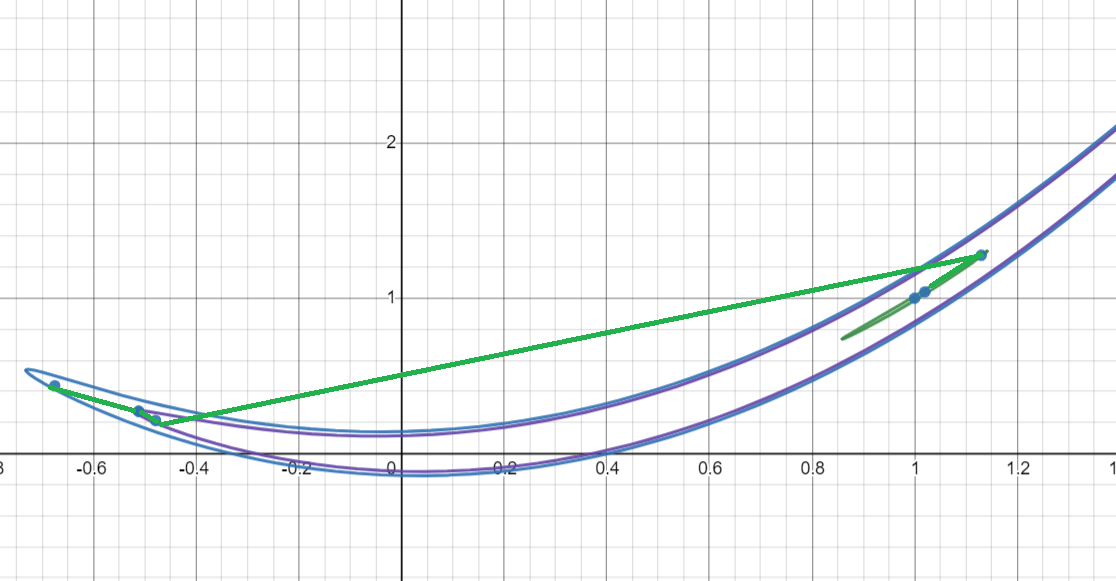
f(x) = 100(x2 - x12)2 + (1 - x1)2



16 итераций

Начальные данные -2 2

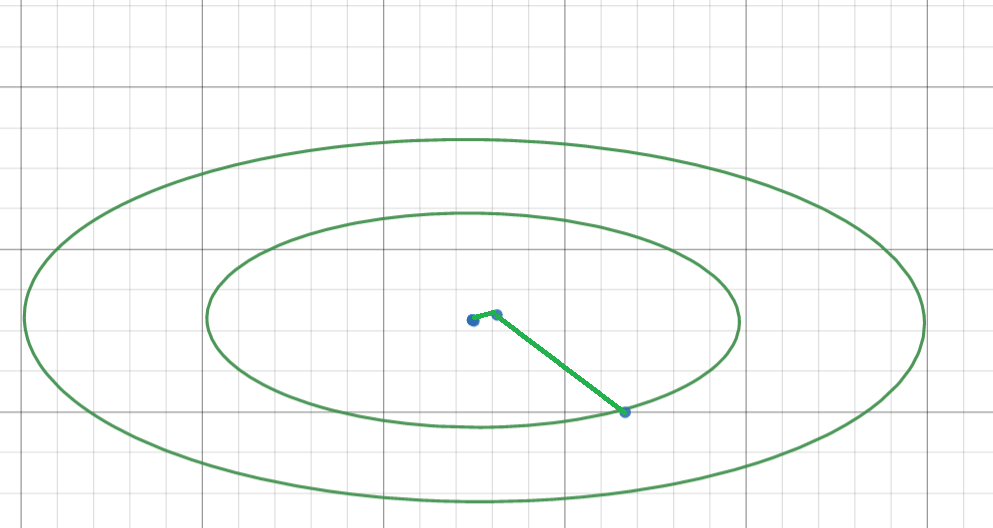
f(x) = 100(x2 - x12)2 + (1 - x1)2\



11 итераций

f(x) = (x2 + x12 - 11)2 + (x1 + x22 - 7)2

5 итераций



Выводы:

1) Метод Ньютона не всегда сходится. Например, классический может проскочить точку минимума из-за большого шага. Модификации сходятся лучше.

2) Метод Ньютона с направлением спуска лучший по скорости сходимости в общем случае. Но его траектория мало отличается от траектории метода с одномерным поиска.

3) Скорость сходимости в классическом методе Ньютона зависит от овражности и может отработать быстрее чем другие.

4) Метод Давидона-Флетчера-Пауэлла и метод Пауэлла делают примерно одинаковое количество итераций.

5) Метод Давидона-Флетчера-Пауэлла и метод Пауэлла могут сходиться за большее число итераций по сравнению с Ньютоновскими.

6) Метод Давидона-Флетчера-Пауэлла и метод Пауэлла требуют меньше вычислений. Метод Пауэлла меньше вычислений чем ДФП.

7) Методы Ньютона (все вариации), Пауэлла, ДФП работают быстрее методов наискорейшего спусков.

Код программы:

#include <iostream>

#include <cmath>

#include <iomanip>

#include <fstream>

#include "LU.h"

#include "Function.h"

ofstream fout("powell\_method1.2Res2.txt");

//===================================================== Операции с векторами и матрицами ================================================================

vector<double> minus\_mult(vector<double> x) {

for (int i = 0; i < x.size(); ++i) {

x[i] = -x[i];

}

return x;

}

vector<double> add(vector<double> x, vector<double> y) {

for (int i = 0; i < x.size(); ++i) {

x[i] += y[i];

}

return x;

}

double norm(vector<double> x) {

double answer = 0;

for (auto val : x) {

answer += val \* val;

}

return sqrt(answer);

}

vector<double> mult\_on\_constant(vector<double> x, double y) {

for (int i = 0; i < x.size(); ++i) {

x[i] \*= y;

}

return x;

}

double dot\_product(vector<double> a, vector<double> b) {

double sum = 0;

for (int i = 0; i < a.size(); ++i) {

sum += a[i] \* b[i];

}

return sum;

}

vector<vector<double>> matrix\_E(int n) {

vector<vector<double>> res(n, vector<double>(n, 0));

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

res[i][i] = 1;

}

return res;

}

vector<vector<double>> vector\_on\_vector(vector<double> a, vector<double> b) {

vector<vector<double>> res(a.size(), vector<double>(a.size(), 0));

for (int i = 0; i < a.size(); ++i) {

for (int j = 0; j < b.size(); ++j) {

res[i][j] = a[i] \* b[j];

}

}

return res;

}

vector<vector<double>> matrix\_minus\_matrix(vector<vector<double>> A, vector<vector<double>> B) {

vector<vector<double>> C(A.size(), vector<double>(A[0].size(), 0));

for (int i = 0; i < A.size(); ++i) {

for (int j = 0; j < A[i].size(); ++j) {

C[i][j] = A[i][j] - B[i][j];

}

}

return C;

}

vector<vector<double>> matrix\_on\_constant(vector<vector<double>> A, double alpha) {

for (int i = 0; i < A.size(); ++i) {

for (int j = 0; j < A[i].size(); ++j) {

A[i][j] \*= alpha;

}

}

return A;

}

vector<double> matrix\_on\_vector(vector<vector<double>> A, vector<double> v) {

vector<double> res;

for (int i = 0; i < A.size(); ++i) {

double sum = 0;

for (int j = 0; j < v.size(); ++j) {

sum += A[i][j] \* v[j];

}

res.push\_back(sum);

}

return res;

}

//============================================================================================================================================

double dychotomy(double a, double b, Function fun,vector<double> x, vector<double> p, double EPS = 1e-8) {

double delta = EPS / 2;

double x1, x2, func\_value1, func\_value2;

while ((b - a) / 2 > EPS) {

x1 = (b + a - delta) / 2;

x2 = (b + a + delta) / 2;

func\_value1 = fun.function(add(x, mult\_on\_constant(p, x1)));

func\_value2 = fun.function(add(x, mult\_on\_constant(p, x2)));

if (func\_value1 <= func\_value2) {

b = x2;

}

else {

a = x1;

}

}

return (a + b) / 2;

}

vector<double> classic\_Newton\_method(vector<double> begin\_value, Function fun, double EPS = 1e-5) {

LU solver;

int counter = 0;

fout << "Iteration 0" << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << '\n';

while (true) {

++counter;

auto antigrad = minus\_mult(fun.gradient(begin\_value));

auto p = solver.solve(fun.hessian(begin\_value), antigrad);

begin\_value = add(begin\_value, p);

fout << "Iteration " << counter << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << '\n';

if (norm(p) <= EPS) {

break;

}

}

return begin\_value;

}

vector <double> one\_dimension\_Newton\_method(vector<double> begin\_value, Function fun, double EPS = 1e-5) {

LU solver;

int counter = 0;

fout << "Iteration 0" << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << '\n';

while (true) {

++counter;

auto antigrad = minus\_mult(fun.gradient(begin\_value));

auto p = solver.solve(fun.hessian(begin\_value), antigrad);

double param = dychotomy(-100, 100, fun, begin\_value, p, EPS);

p = mult\_on\_constant(p, param);

begin\_value = add(begin\_value, p);

fout << "Iteration " << counter << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << "\nParameter = " << param << '\n';

if (norm(p) <= EPS) {

break;

}

}

return begin\_value;

}

vector<double> descent\_dimensional\_Newton\_method(vector<double> begin\_value, Function fun, double EPS = 1e-8) {

LU solver;

int counter = 0;

fout << "Iteration 0" << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << '\n';

auto antigrad = minus\_mult(fun.gradient(begin\_value));

double param = dychotomy(-100, 100, fun, begin\_value, antigrad, EPS);

add(begin\_value, mult\_on\_constant(antigrad, param));

while (true) {

++counter;

auto grad = fun.gradient(begin\_value);

auto s = solver.solve(fun.hessian(begin\_value), minus\_mult(grad));

if (dot\_product(s, grad) < 0) {

antigrad = s;

}

else {

antigrad = minus\_mult(grad);

}

if (norm(s) < EPS) {

break;

}

param = dychotomy(-100, 100, fun, begin\_value, antigrad, EPS);

s = mult\_on\_constant(antigrad, param);

begin\_value = add(begin\_value, s);

fout << "Iteration " << counter << '\n';

for (auto val : begin\_value) {

fout << val << " ";

}

fout << "\nParameter = " << param << '\n';

if (norm(s) < EPS) {

break;

}

}

return begin\_value;

}

vector<double> davidon\_flutcher\_powell\_method(vector<double> begin\_value, Function fun) {

int n = begin\_value.size();

vector<vector<double>> G = matrix\_E(n);

vector<double> w = fun.gradient(begin\_value);

w = minus\_mult(w);

vector<double> x0 = begin\_value;

vector<double> p = w;

double alpha = dychotomy(-10000, 10000, fun, x0, p, 1e-16);

vector<double> x = add(x0, mult\_on\_constant(p, alpha));

vector<double> delta\_x = add(x, minus\_mult(x0));

int counter = 0;

while (norm(delta\_x) >= 1e-8) {

++counter;

vector<double> w1 = minus\_mult(fun.gradient(x));

vector<double> delta\_w = add(w1, minus\_mult(w));

vector<double> v = matrix\_on\_vector(G, delta\_w);

vector<vector<double>> A = matrix\_on\_constant(vector\_on\_vector(delta\_x, delta\_x), 1.0 / dot\_product(delta\_w, delta\_x));

vector<vector<double>> B = matrix\_on\_constant(vector\_on\_vector(v, v), 1.0 / dot\_product(v, delta\_w));

vector<vector<double>> G1 = matrix\_minus\_matrix(G, A);

G1 = matrix\_minus\_matrix(G1, B);

G = G1;

p = matrix\_on\_vector(G1, w1);

alpha = dychotomy(-1000, 1000, fun, x, p, 1e-8);

vector<double> x\_new = add(x, mult\_on\_constant(p, alpha));

vector<double> delta\_x\_new = add(x\_new, minus\_mult(x));

x = x\_new;

delta\_x = delta\_x\_new;

w = w1;

fout << "Iteration " << counter << '\n';

for (auto val : x) {

fout << val << " ";

}

fout << "\nParameter = " << alpha << '\n';

}

fout << counter << '\n';

for (int i = 0; i < x.size(); ++i) {

fout << x[i] << " ";

}

return x;

}

vector<double> powell\_method(vector<double> begin\_value, Function fun) {

int n = begin\_value.size();

vector<vector<double>> G = matrix\_E(n);

auto w = fun.gradient(begin\_value);

w = minus\_mult(w);

vector<double> delta\_w(n);

auto x0 = begin\_value;

auto p = w;

double alpha = dychotomy(-1000, 1000, fun, x0, p, 1e-5);

auto x = add(x0, mult\_on\_constant(p, alpha));

auto delta\_x = add(x, minus\_mult(x0));

int counter = 0;

while (norm(delta\_x) >= 1e-5) {

++counter;

auto w1 = minus\_mult(fun.gradient(x));

delta\_w = add(w1, minus\_mult(w));

auto v = matrix\_on\_vector(G, delta\_w);

auto delta\_wave\_x = delta\_x;

delta\_wave\_x = add(delta\_wave\_x, matrix\_on\_vector(G, delta\_w));

auto A = matrix\_on\_constant(vector\_on\_vector(delta\_wave\_x, delta\_wave\_x), 1.0 / dot\_product(delta\_w, delta\_wave\_x));

auto G1 = matrix\_minus\_matrix(G, A);

G1 = G;

p = matrix\_on\_vector(G1, w1);

alpha = dychotomy(-1000, 1000, fun, x, p, 1e-5);

auto x\_new = add(x, mult\_on\_constant(p, alpha));

auto delta\_x\_new = add(x\_new, minus\_mult(x));

x = x\_new;

delta\_x = delta\_x\_new;

w = w1;

fout << "Iteration " << counter << '\n';

for (auto val : x) {

fout << val << " ";

}

fout << "\nParameter = " << alpha << '\n';

}

fout << counter << '\n';

for (int i = 0; i < x.size(); ++i) {

fout << x[i] << " ";

}

return x;

}

int main()

{

Function fun;

fout << fixed << setprecision(10);

fun.function = ([](vector<double> x1) {

auto x = x1[0];

auto y = x1[1];

auto z = x1[2];

auto r = x1[3];

return pow(x + 10 \* y, 2) + 5 \* pow(z - r, 2) + pow(y - 2 \* z, 4) + 10\* pow(x - r, 4);

});

powell\_method({-2, 2}, fun);

}