

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО  
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ  
ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

**Отчет по лабораторной работе №4**

Работу выполнили:

Гранкин Максим, группа М3237  
Панов Иван, группа М3239  
Назаров Георгий

Преподаватель: Корсун М.М.

Санкт-Петербург  
2021 г.

# 1. Введение

## 1.1. Постановка задачи

- Реализовать различные методы Ньютона: классический, с одномерным спуском, с направлением спуска.
- Исследовать работу методов на функциях с заданным начальным приближением. Сравнить результаты с методом наискорейшего спуска. Выявить наилучший метод Ньютона.
- Реализовать квазиньютоновские методы: Давидона-Флетчера-Пауэлла и метод Пауэлла. Провести сравнение с наилучшим методом Ньютона.
- Провести исследование и сделать выводы о каждом методе.

# 2. Теоретическое обоснование

## 2.1. Методы Ньютона

Метод Ньютона относится к классу итерационных методов. Если целевая функция  $f(x) \in E^n$  дважды дифференцируема в пространстве  $E^n$ , то в процессе минимизации можно использовать Гессиан функции. За счет использования Гессиана повышается скорость сходимости итерационной последовательности.

На каждой итерации происходит квадратичная аппроксимация целевой функции в некоторой окрестности точки. Для текущей точки в итерационном процессе происходит аппроксимация квадратичной функции, однако следующей точку релаксационной последовательности определяется как точка минимума аппроксимирующей квадратичной функции описанной выше. Аппроксимация основана на формуле Тейлора.

Пренебрегая остаточным членом в форме Пеано запишем формулу Тейлора и обозначим за  $\phi_k(x)$

$$\phi_k(x) = f(x_{k-1}) + \langle \nabla f(x_{k-1}), x - x_{k-1} \rangle + \frac{1}{2} \langle H(x_k)(x - x_{k-1}), x - x_{k-1} \rangle$$

Функция  $\phi_k(x)$  имеет одну точку в которой достигается минимум.

Условие минимума:

$$\nabla \phi_k(x) = \nabla f(x_{k-1}) + H(x_{k-1})(x - x_{k-1}) = 0$$

Тогда точка релаксационной последовательности определяется как:

$$x_k = x_{k-1} - H^{-1}(x_{k-1}) \cdot \nabla f(x_{k-1}), k \in N$$

$p_k$  - направление спуска.

Если  $H(x_{k-1})$  положительно определенная, то  $p_k$  принято называть ньютоновским направлением спуска.

В классическом варианте  $p_k = -H^{-1}(x_{k-1})\nabla f(x_{k-1})$

### 2.1.1. Классический

Алгоритм минимизации:

1. Рассчитать градиент функции в точке  $x_k$  и рассчитать Гессиан функции в точке  $x_k$
2. Решить систему линейных алгебраических уравнений:  $H \cdot p_k = -\nabla f(x)$
3. Произвести пересчет  $x_k = x_{k-1} + p_k$
4. Минимум найден, если  $\|x_k - x_{k-1}\| < eps$ . Это эквивалентно  $\|p_k\| < eps$ .  
Иначе, повторить алгоритм с пункта 1.

Классический метод Ньютона не обладает свойством глобальной сходимости, так как если начальная точка  $x_0$  достаточно далека от  $x_*$ , то метод не сходится.

Имеет квадратичную локальную сходимость при условии того, что Гессиан функции удовлетворяет в окрестности  $x_*$  условию Липшица. Таким образом, в малой окрестности  $x_*$  метод будет сходится значительно быстрее.

### 2.1.2. С одномерным спуском

Точку минимума  $\phi_k(x)$  можно считать вспомогательным приближением.

$\tilde{x}_k$  - вспомогательная точка для построения релаксационной последовательности  $\{x_k\}$

Тогда точку релаксационной последовательности можно определить как:

$x_k = x_{k-1} + \alpha_k \cdot (\tilde{x}_k - x_{k-1}) = x_{k-1} + \alpha_k \cdot p_k$ , где  $\alpha_k = \min_\alpha f(x_{k-1} + \alpha \cdot p_k)$  и  $p_k$  это направление спуска.

Алгоритм минимизации аналогичен классическому, однако  $x_k$  определяется как:  $x_k = x_{k-1} + \alpha_k \cdot p_k$

Эффективность методы существенно зависит от того, является ли  $p_k$  направлением спуска.

Метод обладает свойством глобальной сходимости.

### 2.1.3. С направлением спуска

Данный метод позволяет избавится от ошибочного выбора направления возрастания функции. Ошибочный выбор связан с точками максимума функции и седловыми точками. В этих случаях необходимо проверять является ли  $p_k$  направлением спуска.

$p_k$  - направление спуска, то  $(p_k)^T \cdot \nabla f(x_k) < 0$

Если  $(p_k)^T \cdot \nabla f(x_k) < 0$ , то  $p_k$  не является направлением спуска. В этом случае следует использовать антиградиент.

Тогда зададим  $p_k$  следующим образом:

$$p_k = \begin{cases} p_k, (p_k)^T \cdot \nabla f(x_k) < 0 \\ -\nabla f(x_k), (p_k)^T \cdot \nabla f(x_k) > 0 \end{cases}$$

Для поиска оптимального  $a_k$  можно использовать одномерный поиск. Остальные шаги аналогичны предыдущим методам.

Метод обладает свойством глобальной сходимости.

## 2.2. Квазиньютоновские методы

Квазиньютоновские методы объединяют в себе достоинства метода Ньютона и метода наискорейшего спуска. Они не требуют обращений к Гессиану, однако при этом сохраняется высокая сходимость итерационной последовательности.

Общий вид релаксационной последовательности:

$$x_k = x_{k-1} + \alpha_k \cdot p_k$$

$p_k$  - направление спуска. Определяется как:  $p_k = G_k \cdot w_k, k \in N$

$w_k = -\nabla f(x_{k-1})$  - антиградиент функции  $G_k$  - положительно определенная матрица порядка  $(n \times n)$  специального вида для каждого квазиньютоновского метода

Вычисление матриц  $G_k$

$G_{k \rightarrow \infty} \rightarrow H^{-1}(x_*)$  - матрица  $G_k$  при достаточно больших  $k$  должна сходить к обратной матрице Гессе.  $x_*$  точка минимума целевой функции.

На первой итерации матрица  $G_k$  является единичной, таким образом направление  $p_k$  равняется антиградиенту.

За счет того, что матрица  $G_k$  на завершающей стадии поиска близка к  $H^{-1}(x_{k-1})$  мы можем ожидать высокую сходимость итерационной последовательности, присущую методу Ньютона.

Параметр  $\alpha_k$  чаще всего выбирается как исчерпывающий спуск в направлении  $r_k$ , однако возможно задавать  $\alpha_k = 1$  или применять дробление шага.

Основная идея квазиньютоновских методов заключается в том, что специальная матрица  $G_k$  должна быть достаточно близка к  $H^{-1}(x_{k-1})$  и вычисление происходит за счет аппроксимации  $H^{-1}(x_{k-1})$ . Выбор удачной аппроксимации позволяет существенно сократить объем вычислений по сравнению с обращением к Гессиану, таким образом упрощается процедура построения направления спуска  $p_k$ .

Различные квазиньютоновские методы определяет выбор "поправочной" матрицы  $\Delta G_k$ , используемой для вычисления  $G_{k+1}$ .

Квазиньютоновские методы обладают глобальной сходимостью только в случае применения рестартов.

Имеют сверхлинейную скорость сходимости, однако если Гессиан функции удовлетворяет условию Липшица, то в результате будет достигнута квадратичная скорость.

### **2.2.1. Метод Давидона-Флэтчера-Пауэлла**

Данный метод обладает рядом свойств, опишем их подробнее:

1.  $G_k$  сохраняет положительную определенность: если  $G_k > 0$ , то и  $G_{k+1} > 0$ , при условии  $(\Delta x_k)^T \Delta w_k > 0$
2. Если  $G_k$  симметричная, то и  $G_{k+1}$  симметричная
3. При минимизации квадратичной функции с положительно определенной матрицей  $A$  метод сводится к методу сопряженных направлений. Следовательно, метод дает точное решение не более чем за  $n$  итераций.
4. Матрицы  $G_k$  вычисленные по методу связаны равенством:  

$$G_k \cdot A \cdot p_i = p_i.$$
 При  $k = n$ ,  $n$  векторов  $p_i$  являются собственными для симметричной матрицы  $G_n \cdot A$ , а собственные значения равны единице. Таким образом,  $G_n$  оказывается обратной к Гессиану квадратичной функции.
5. Если целевая функция не квадратичная, то метод не позволяет найти минимум за конечное число итераций. Для уменьшения ошибки и вероятности появления линейно зависимых направлений спуска, принято каждые  $n$  итераций принимать  $G_n = I$  (единичной матрице)
6. Если целевая функция является квадратичной с положительно определенным Гессианом, то метод после  $n$  итераций дает:  

$$H^{-1} = \sum_{i=1}^n n \frac{\Delta x_i (\Delta x_i)^T}{(\Delta x_i)^T \Delta w_i}$$

Алгоритм минимизации:

$G_1 = I$  - единичная матрица.

$w_1 = -\nabla f(x_0)$

$p_1 = w_1$

$\alpha_1 = \min_\alpha f(x_0 + \alpha p_1)$

$x_1 = x_0 + \alpha_1 p_1$

$\Delta x_1 = x_1 - x_0$

$\forall k > 1$

$w_k = -\nabla f(x_{k-1})$

$\Delta w_k = w_k - w_{k-1}$

$v_k = G_{k-1} \Delta w_k$

$G_k = G_{k-1} - \frac{\Delta x_{k-1}(\Delta x_{k-1})^T}{\langle \Delta w_k, \Delta x_{k-1} \rangle} - \frac{v_k(v_k)^T}{\langle v_k, \Delta w_k \rangle}$

$p_k = G_k \cdot w_k$

$\alpha_k = \min_\alpha f(x_{k-1} + \alpha_k \cdot p_k)$

$x_k = x_{k-1} + \alpha_k \cdot p_k$

$\Delta x_k = x_k - x_{k-1}$

Условие остановки:  $\|\Delta x_k\| < eps$

Метод крайней чувствителен к точности одномерного поиска.

### 2.2.2. Метод Пауэлла

Аналогичен методу ДФП, однако имеет другое реккурентное соотношение для вычисления матриц  $G_k$ :

$G_{k+1} = G_k - \frac{\Delta \tilde{x}_k (\Delta \tilde{x}_k)^T}{\langle \Delta w_k, \Delta \tilde{x}_k \rangle}$

### 3. Исследование методов Ньютона на различных функциях

Проведем исследование на двух функциях:  $\sin(x)$  и  $8 \cdot x^2 + 4 \cdot x \cdot y + 5 \cdot y^2$ , а также на различных начальных приближениях.

Точность:  $\text{eps} = 10^{-5}$

Для поиска ньютоновского направления используется метод Гаусса.

#### 3.1. Классический метод Ньютона

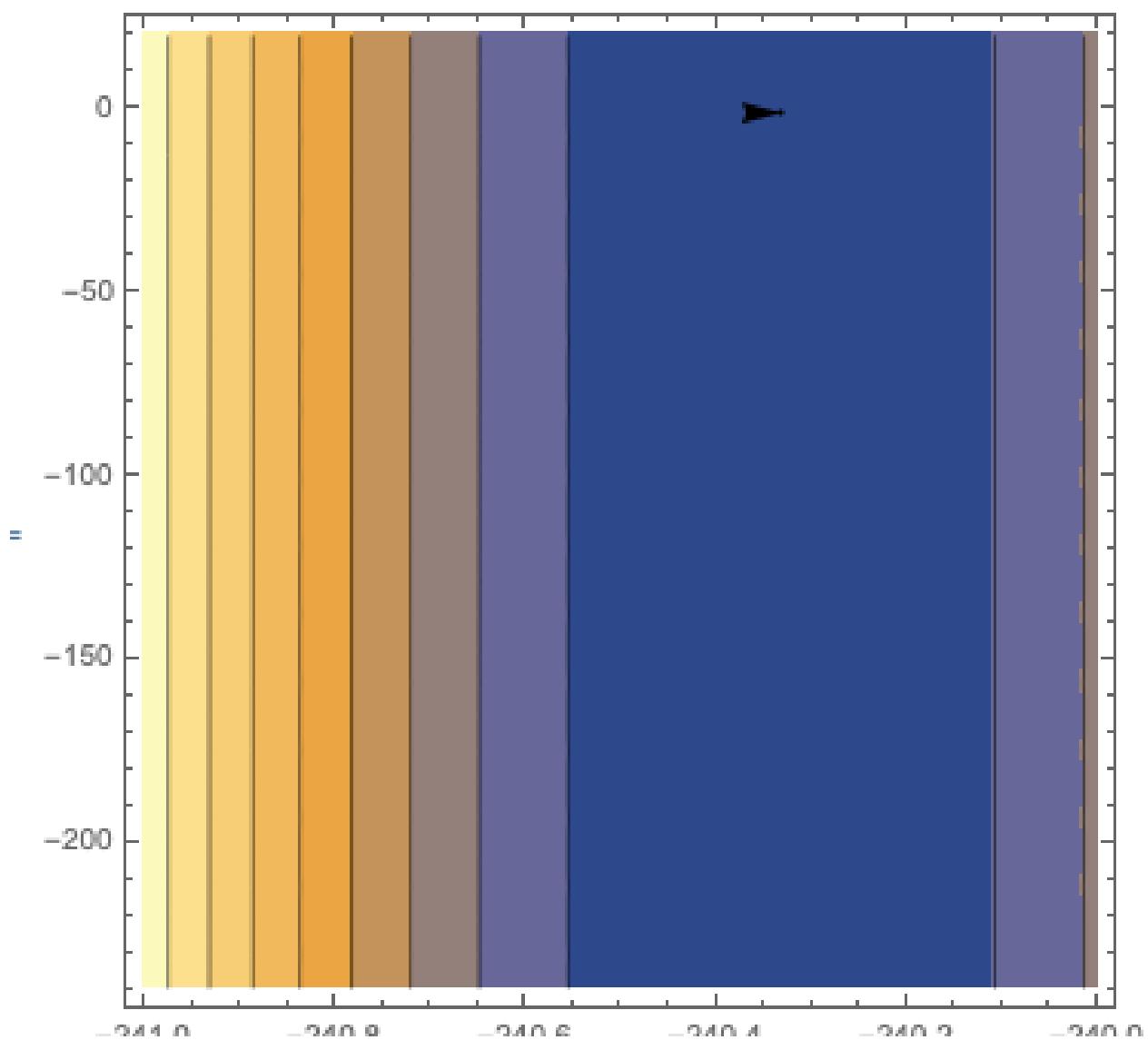
##### 3.1.1. Функция $\sin(x)$

Начальное приближение:  $x = 1$ .

Количество итераций: 4.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	1.0000000	1.6420840	0.6420840	0.9974601
1	1.0000000	1.5706701	-0.0714138	1.0000000
2	1.0000000	1.5707913	0.0001212	1.0000000
3	1.0000000	1.5707913	0.0000000	1.0000000

Как можно заметить, метод нашел точку максимума, вместо минимума. Это связано с тем, что для  $\sin(x)$  Гессиан отрицательно определен.



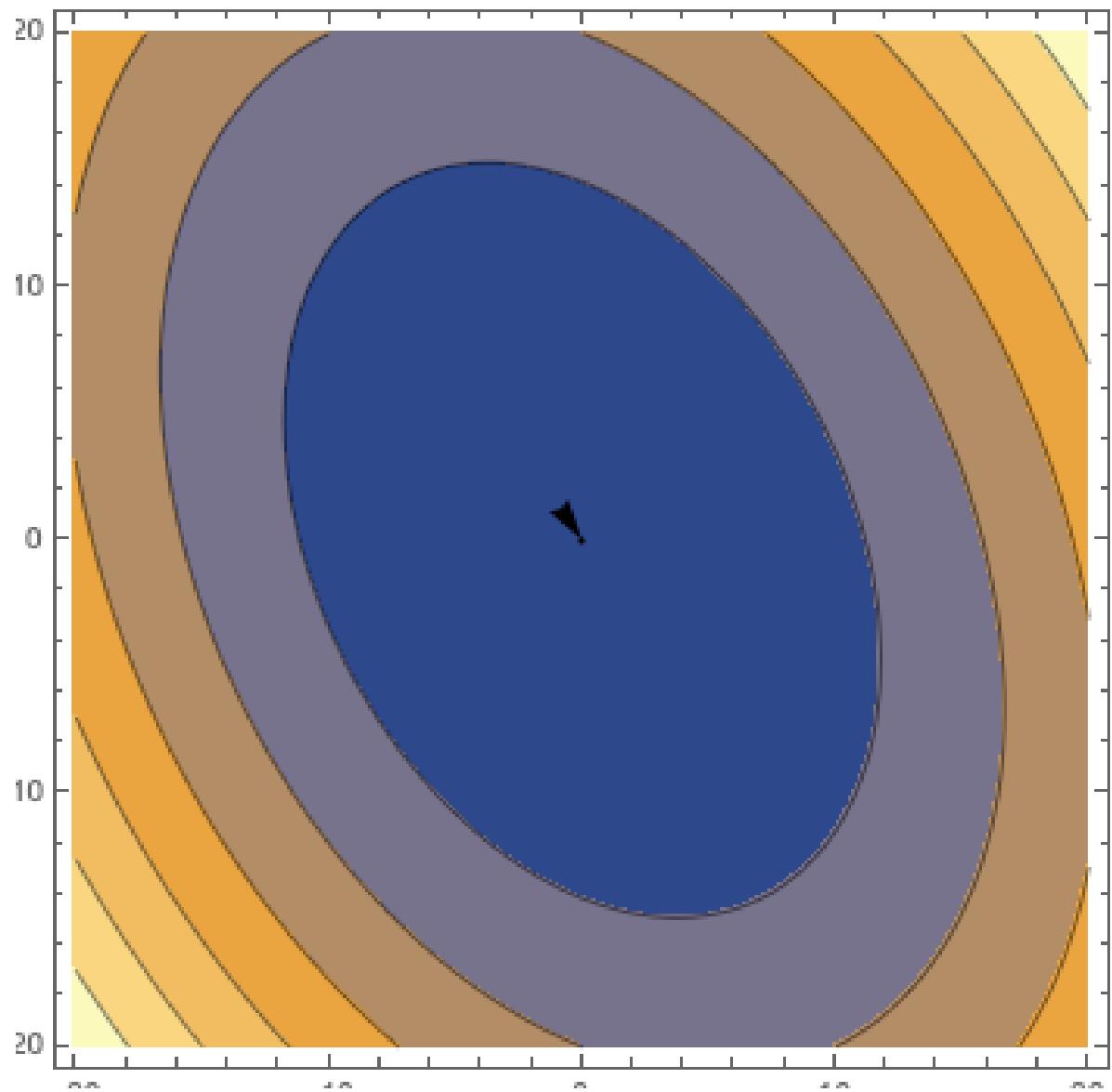
Начальное приближение:  $x = -\frac{\pi}{2} + 0.03$   
Количество итераций: 2

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	1.0000000	-1.5708103	-0.0300140	-1.0000000
1	1.0000000	-1.5708013	0.0000090	-1.0000000

При данном начальном приближении метод находит истинный минимум, Гессиан определен положительно. Начальное приближение находится в маленькой окрестности минимума.

Начальное приближение:  $x = -\pi + 0.03$   
Количество итераций: 4

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	1.0000000	30.2006548	33.3122475	-0.9374640
1	1.0000000	29.8293463	-0.3713085	-0.9998754
2	1.0000000	29.8451265	0.0157802	-1.0000000
3	1.0000000	29.8451252	-0.0000013	-1.0000000



### 3.1.2. Функция: $8 \cdot x^2 + 4 \cdot x \cdot y + 5 \cdot y^2$

Начальное приближение:  $(x, y) = (1.0, 2.0)$

Количество итераций: 3

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0000000	-0.0000110	0.0000067	-1.0000110	-1.9999933	0.0000000
1	1.0000000	-0.0000042	-0.0000033	0.0000068	-0.0000100	0.0000000
2	1.0000000	-0.0000042	-0.0000033	0.0000000	-0.0000000	0.0000000

Начальное приближение:  $(x, y) = (2.0, 1.0)$

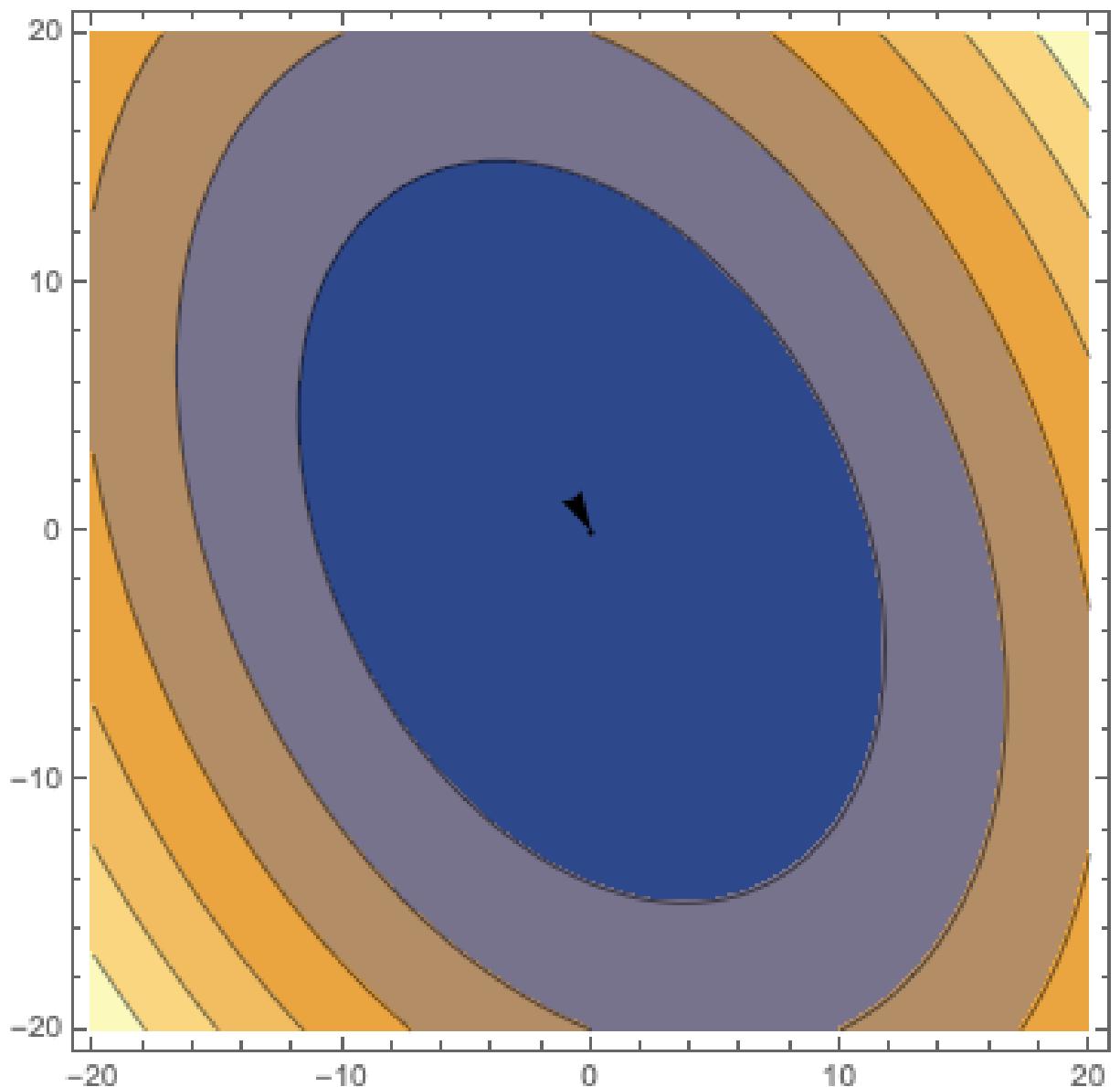
Количество итераций: 3

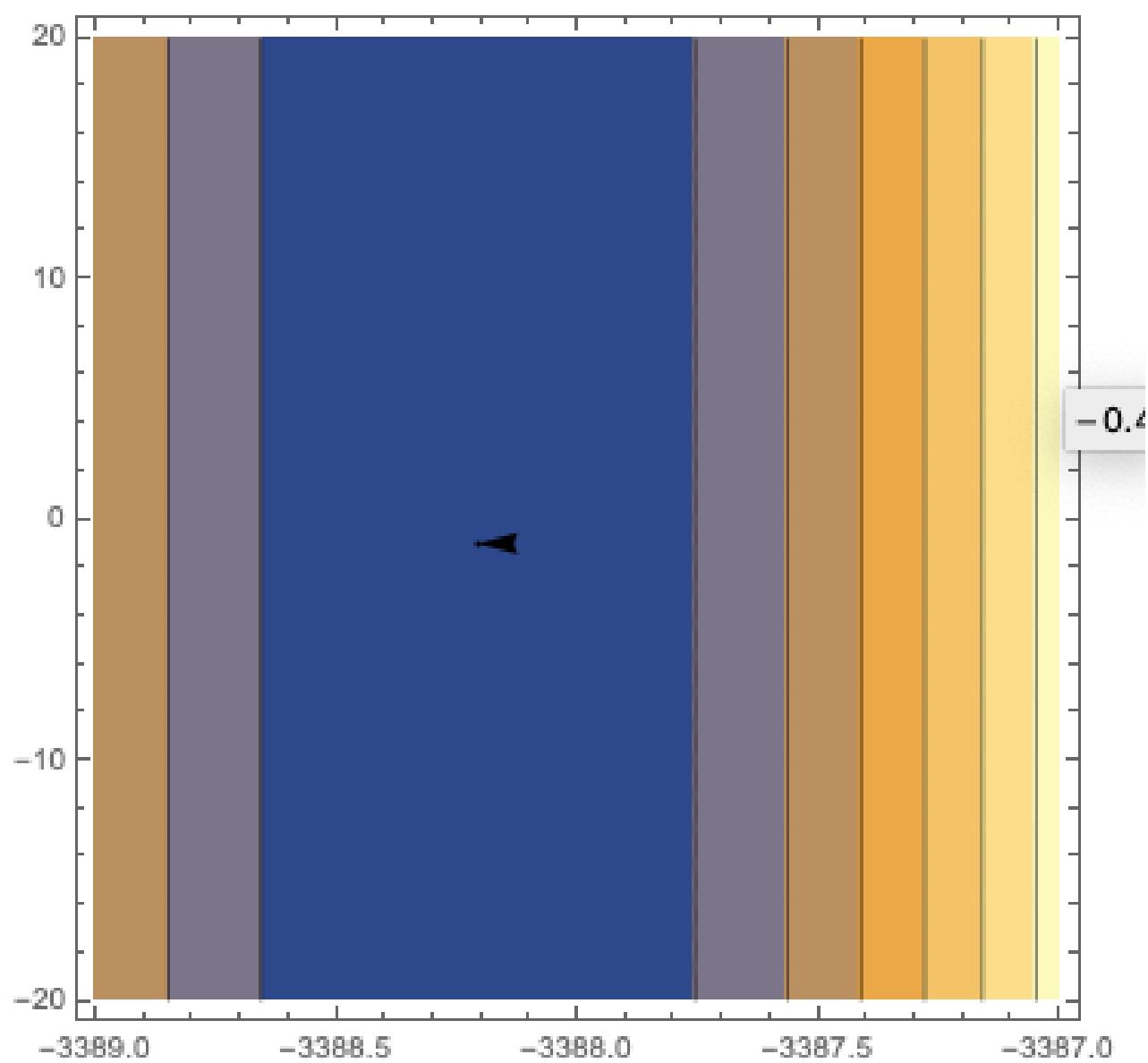
Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0000000	-0.0000060	0.0000224	-2.0000060	-0.9999776	0.0000000
1	1.0000000	-0.0000042	-0.0000033	0.0000018	-0.0000257	0.0000000
2	1.0000000	-0.0000042	-0.0000033	0.0000000	-0.0000000	0.0000000

Начальное приближение:  $(x, y) = (0.0, 0.0)$

Количество итераций: 1

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0000000	-0.0000042	-0.0000033	-0.0000042	-0.0000033	0.0000000





### 3.2. Метод Ньютона с одномерной оптимизацией методом золотого сечения

#### 3.2.1. Функция $\sin(x)$

Начальное приближение:  $x = 1$ .

Количество итераций: 2.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	-346.5002993	-221.4822838	0.6420840	-1.0000000
1	-0.5184478	-221.4822821	-0.0000033	-1.0000000

Начальное приближение:  $x = -\frac{\pi}{2} + 0.03$

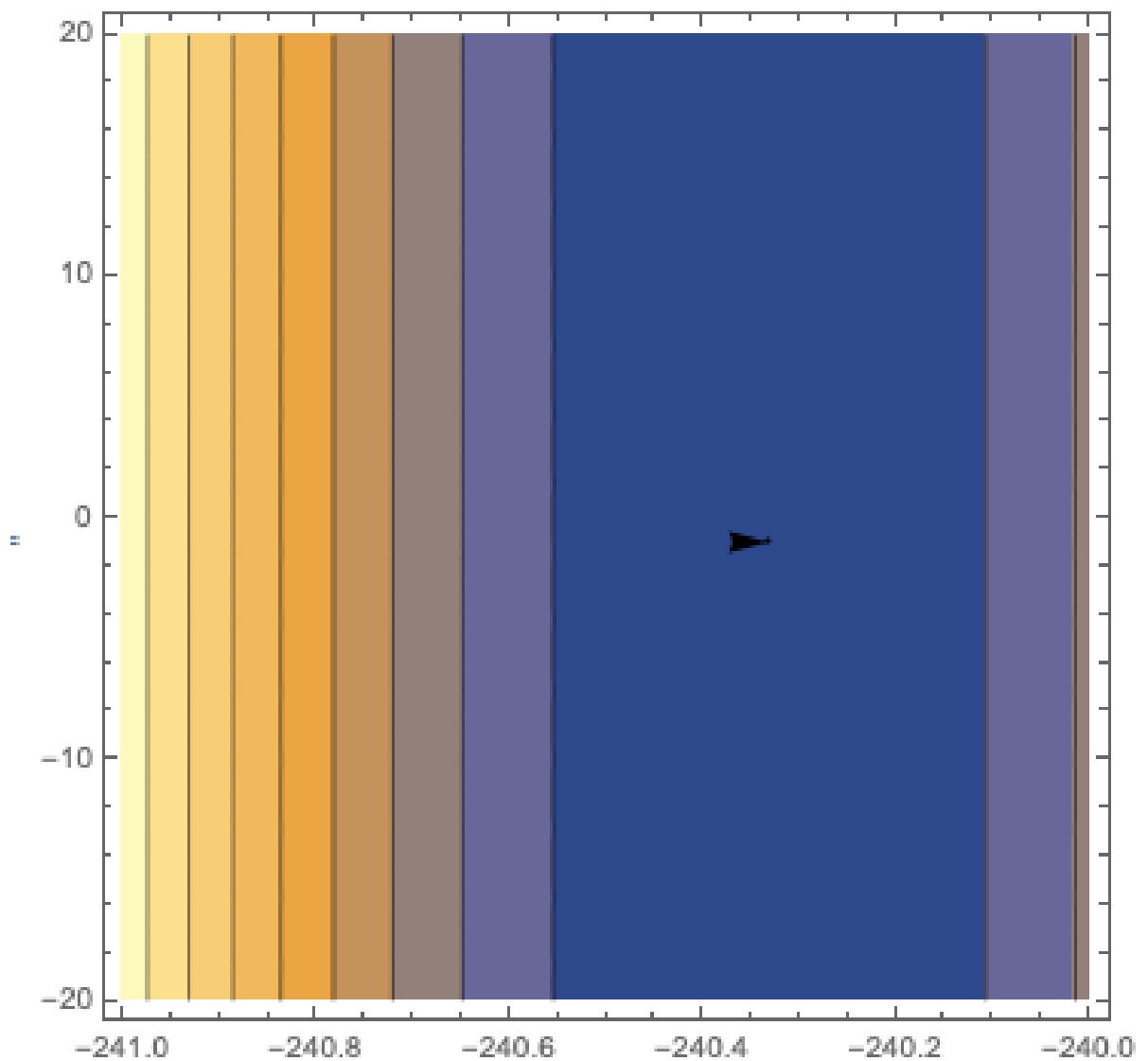
Количество итераций: 2.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	210.3414161	-7.8539817	-0.0300140	-1.0000000
1	-0.0091814	-7.8539816	-0.0000049	-1.0000000

Начальное приближение:  $x = -\pi + 0.03$

Количество итераций: 3.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	-708.2025967	-23594.9317404	33.3122475	-1.0000000
1	1.0452909	-23594.9316248	0.0001106	-1.0000000
2	0.0041431	-23594.9316248	-0.0000050	-1.0000000



### 3.2.2. Функция: $8 \cdot x^2 + 4 \cdot x \cdot y + 5 \cdot y^2$

Начальное приближение:  $(x, y) = (1.0, 2.0)$

Количество итераций: 3

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999990	-0.0000100	0.0000088	-1.0000110	-1.9999933	0.0000000
1	0.9014830	-0.0000047	-0.0000021	0.0000058	-0.0000121	0.0000000
2	-0.0000087	-0.0000047	-0.0000021	0.0000006	-0.0000012	0.0000000

Начальное приближение:  $(x, y) = (2.0, 1.0)$

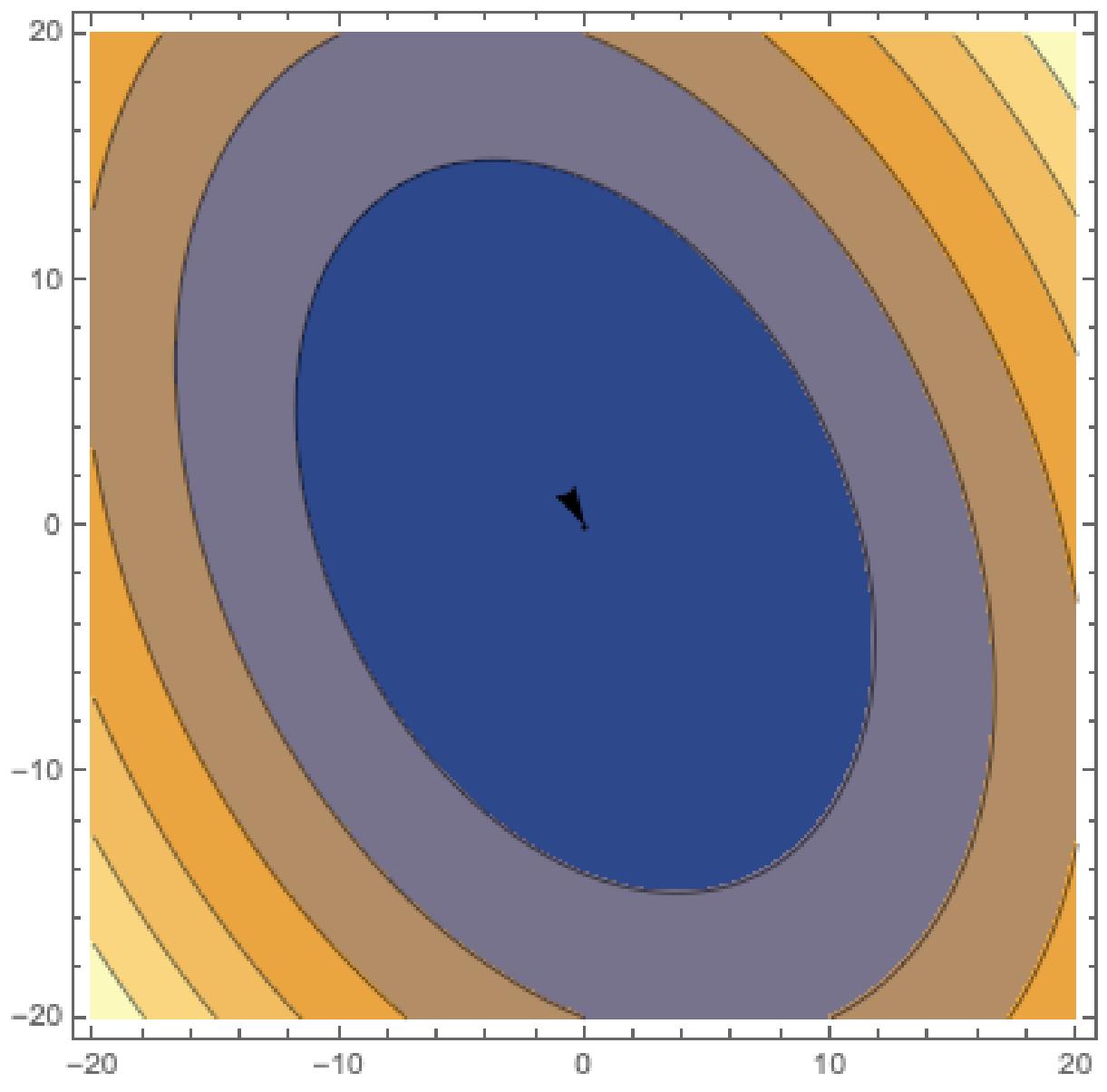
Количество итераций: 3

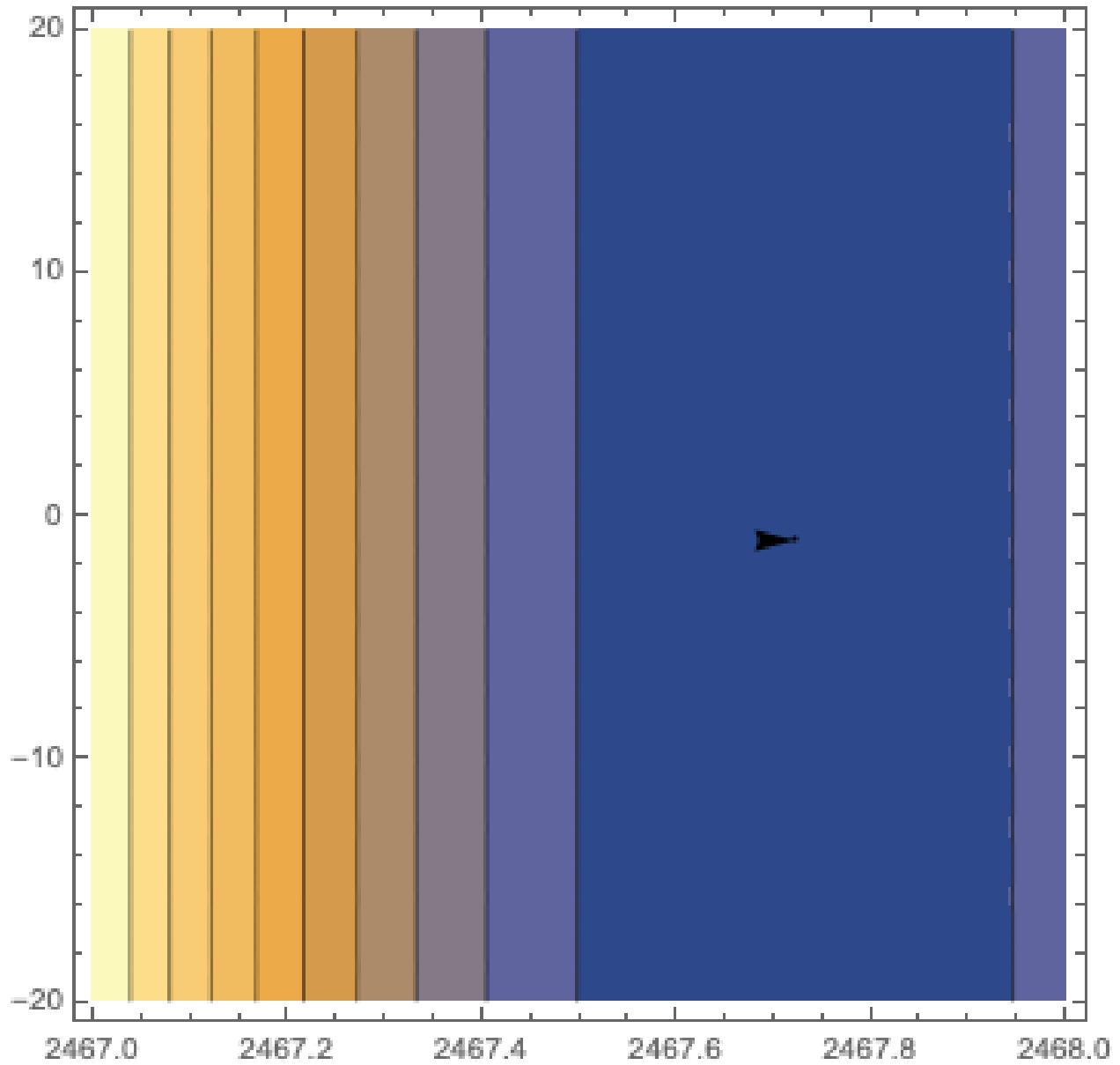
Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999990	-0.0000039	0.0000234	-2.0000060	-0.9999776	0.0000000
1	0.8118490	-0.0000041	0.0000017	-0.0000003	-0.0000268	0.0000000
2	-0.0000141	-0.0000041	0.0000017	-0.0000000	-0.0000050	0.0000000

Начальное приближение:  $(x, y) = (0.0, 0.0)$

Количество итераций: 1

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	-0.0000000	0.0000000	0.0000000	-0.0000042	-0.0000033	0.0000000





### 3.3. Метод ньютона с направлением спуска

В качестве одномерной оптимизации также выбран метод золотого сечения.

#### 3.3.1. Функция $\sin(x)$

Начальное приближение:  $x = 1$ .

Количество итераций: 2.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	528.0679989	-284.3141357	-0.5402981	-1.0000000
1	-0.1219434	-284.3141352	-0.0000044	-1.0000000

Начальное приближение:  $x = -\frac{\pi}{2} + 0.03$

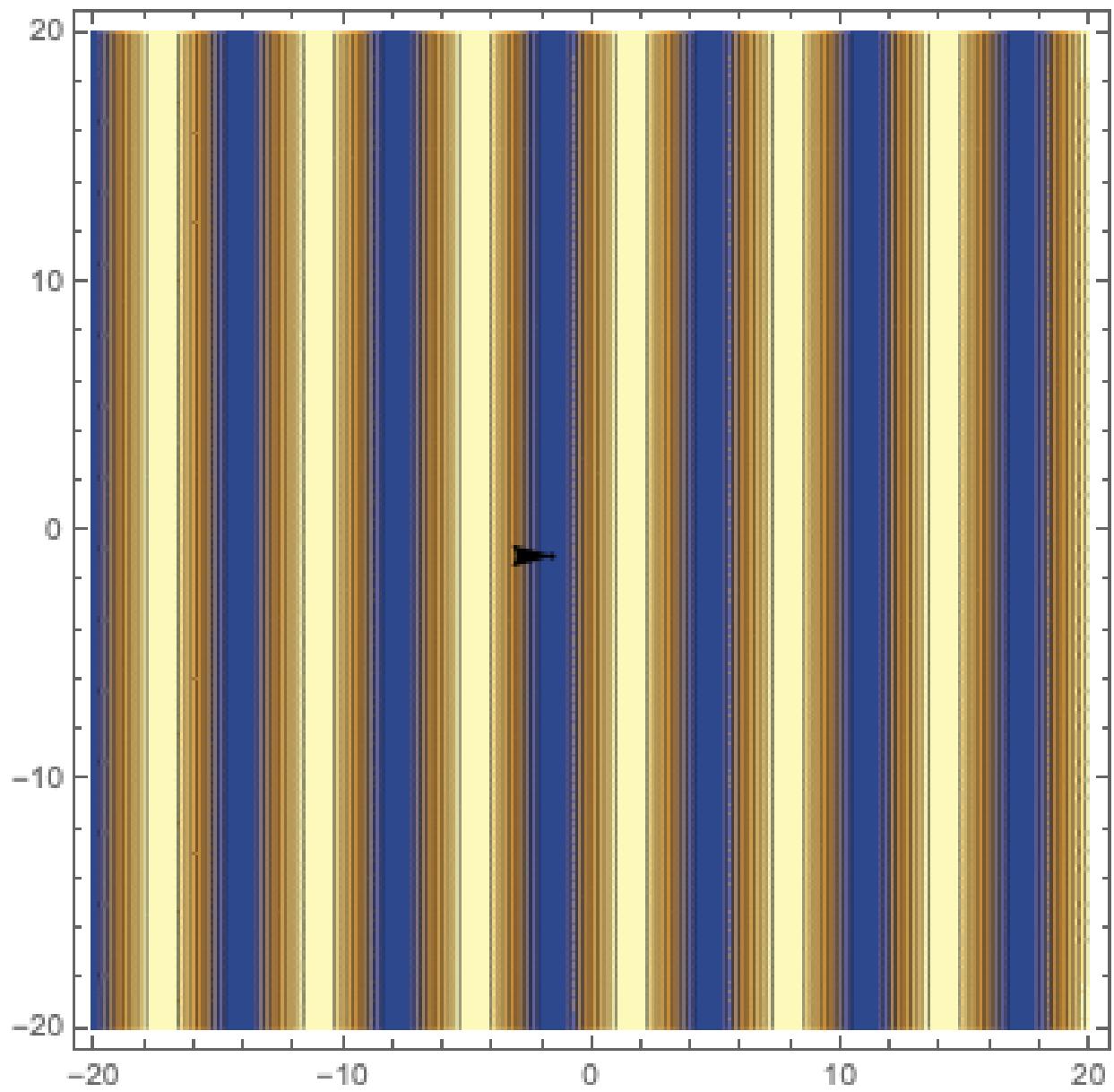
Количество итераций: 2.

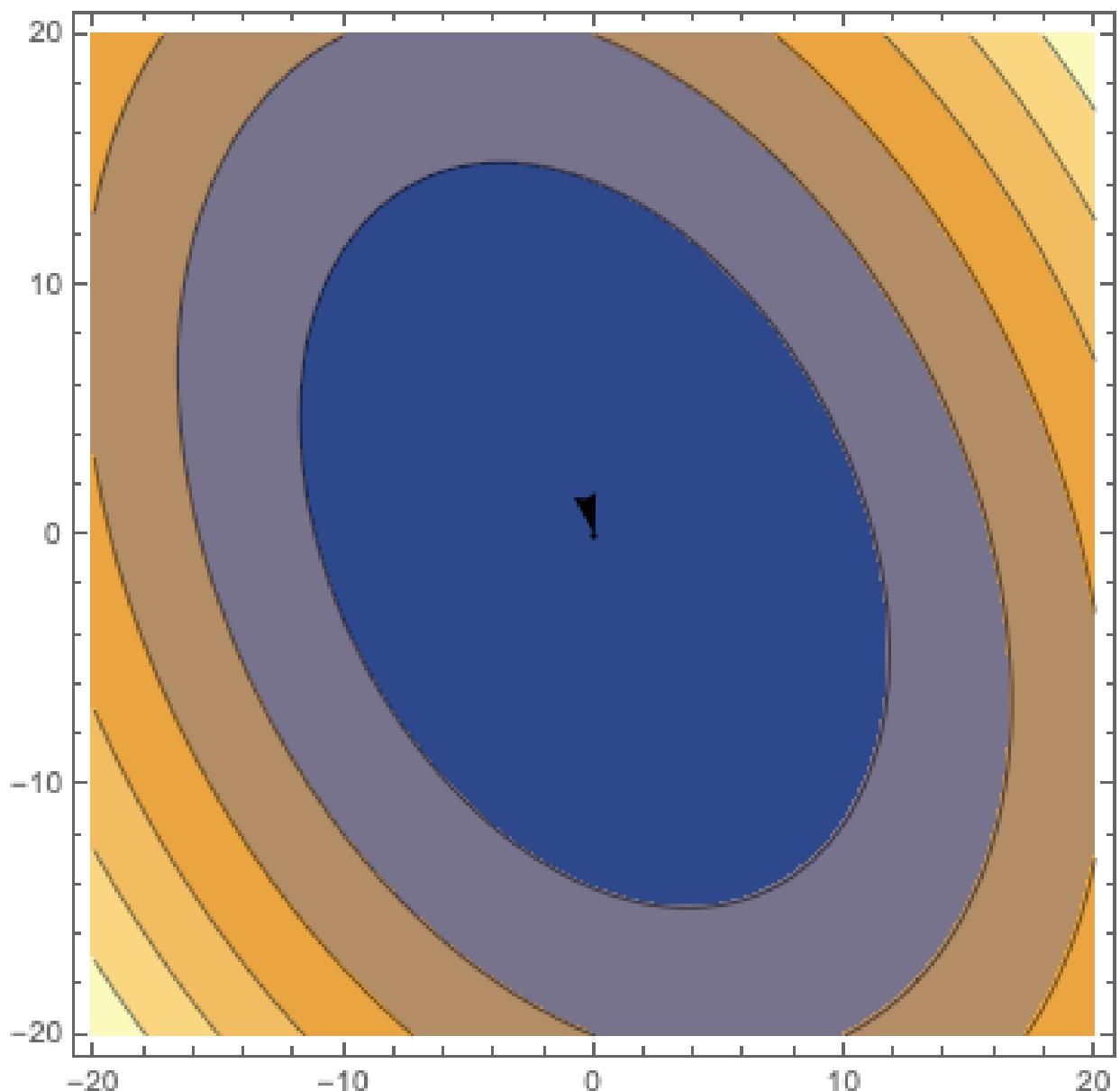
Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	210.3414161	-7.8539817	-0.0300140	-1.0000000
1	-0.0091814	-7.8539816	-0.0000049	-1.0000000

Начальное приближение:  $x = -\pi + 0.03$

Количество итераций: 3.

Итерация	$\alpha_k$	$x_k$	$p_k$	$f(x_k)$
0	-708.2025967	-23594.9317404	33.3122475	-1.0000000
1	1.0452909	-23594.9316248	0.0001106	-1.0000000
2	0.0041431	-23594.9316248	-0.0000050	-1.0000000





### 3.3.2. Функция: $8 \cdot x^2 + 4 \cdot x \cdot y + 5 \cdot y^2$

Начальное приближение:  $(x, y) = (1.0, 2.0)$

Количество итераций: 2

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999990	-0.0000100	0.0000088	-1.0000110	-1.9999933	0.0000000
1	0.9014830	-0.0000047	-0.0000021	0.0000058	-0.0000121	0.0000000
2	-0.0000087	-0.0000047	-0.0000021	0.0000006	-0.0000012	0.0000000

Начальное приближение:  $(x, y) = (2.0, 1.0)$

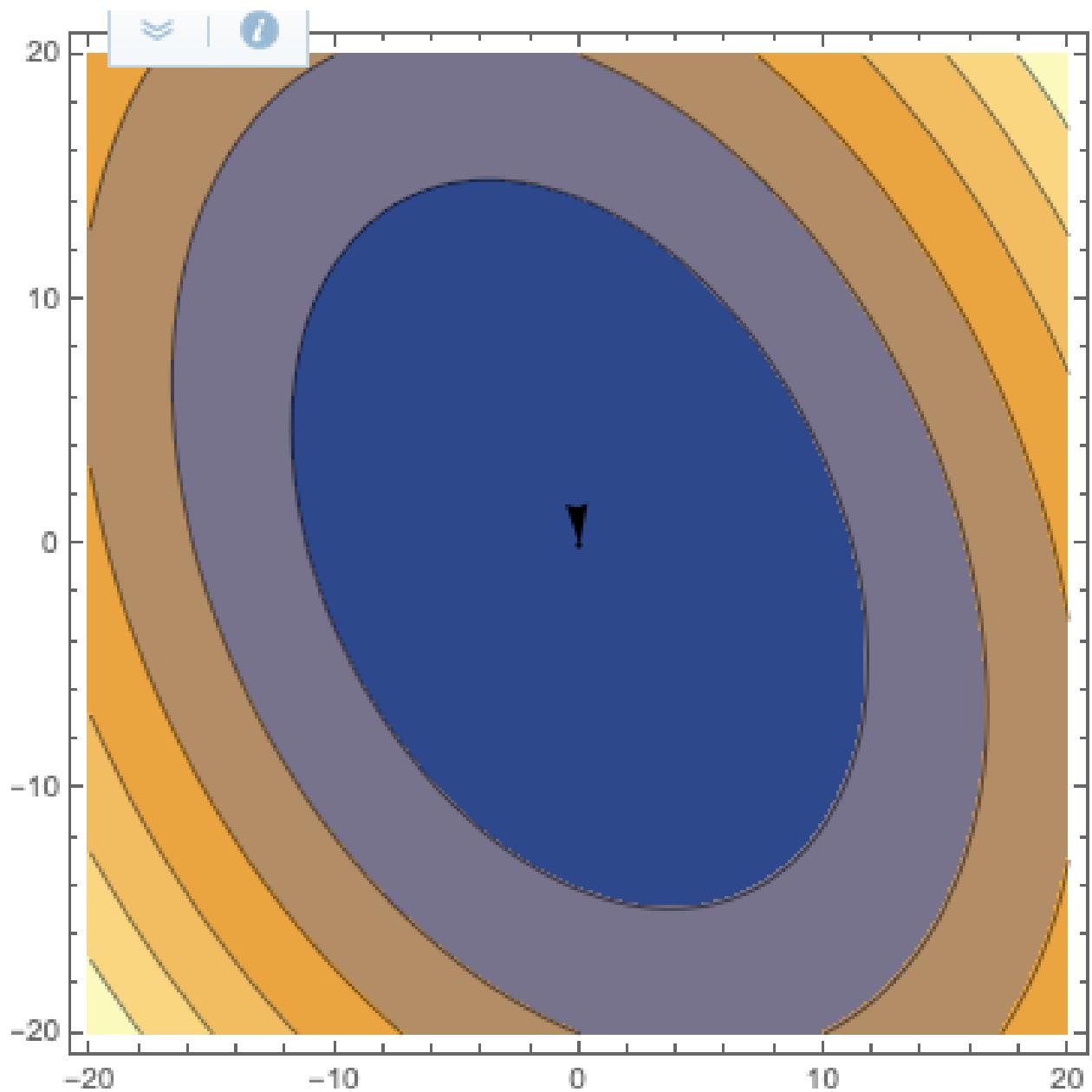
Количество итераций: 3

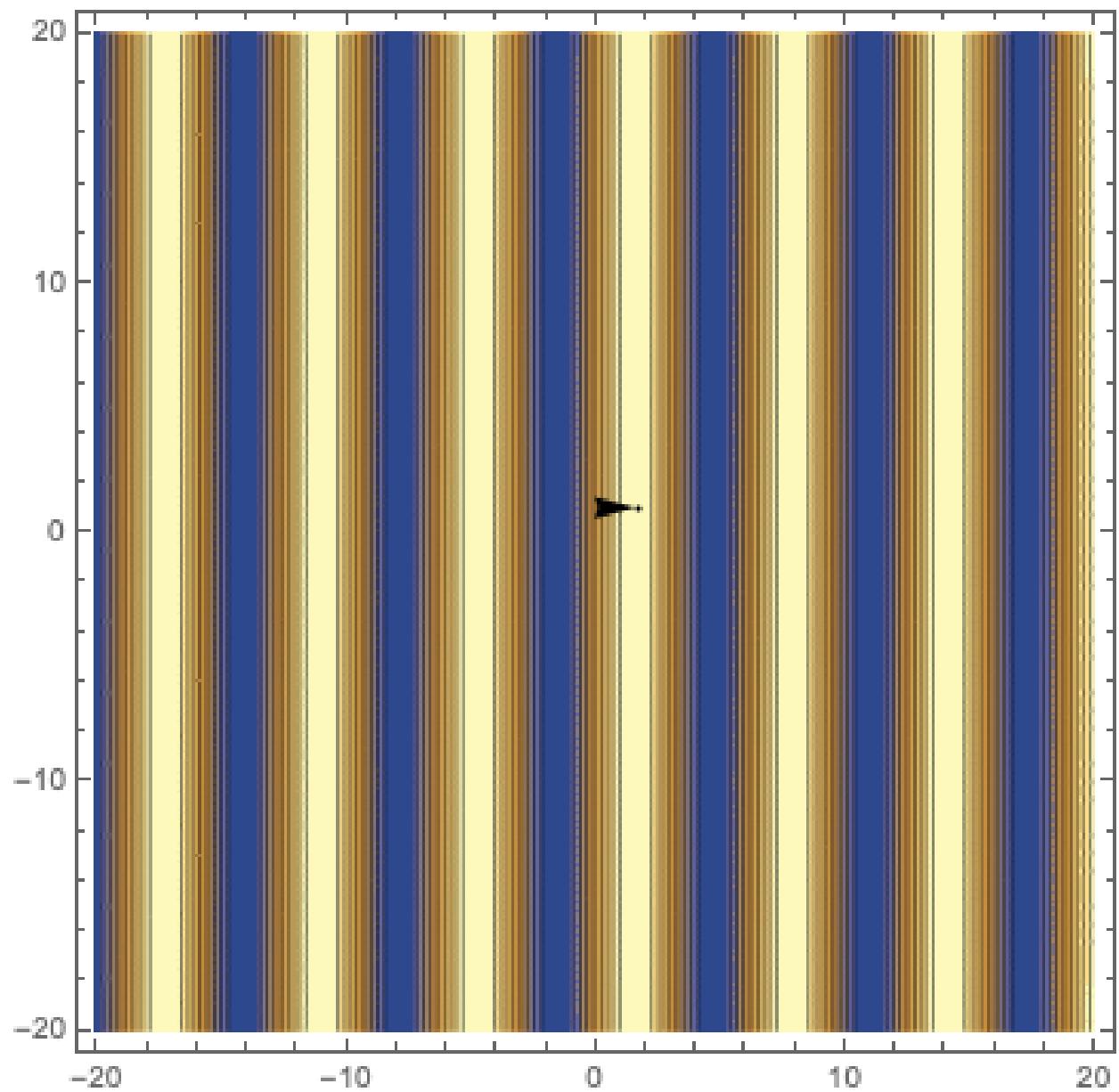
Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999990	-0.0000039	0.0000234	-2.0000060	-0.9999776	0.0000000
1	0.8118490	-0.0000041	0.0000017	-0.0000003	-0.0000268	0.0000000
2	-0.0000141	-0.0000041	0.0000017	-0.0000000	-0.0000050	0.0000000

Начальное приближение:  $(x, y) = (0.0, 0.0)$

Количество итераций: 1

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	-0.0000000	0.0000000	0.0000000	-0.0000042	-0.0000033	0.0000000





## 4. Исследование работы методов на заданных функциях

Функции:

$$f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2 \text{ где } x_0 = (4, 1)^T$$

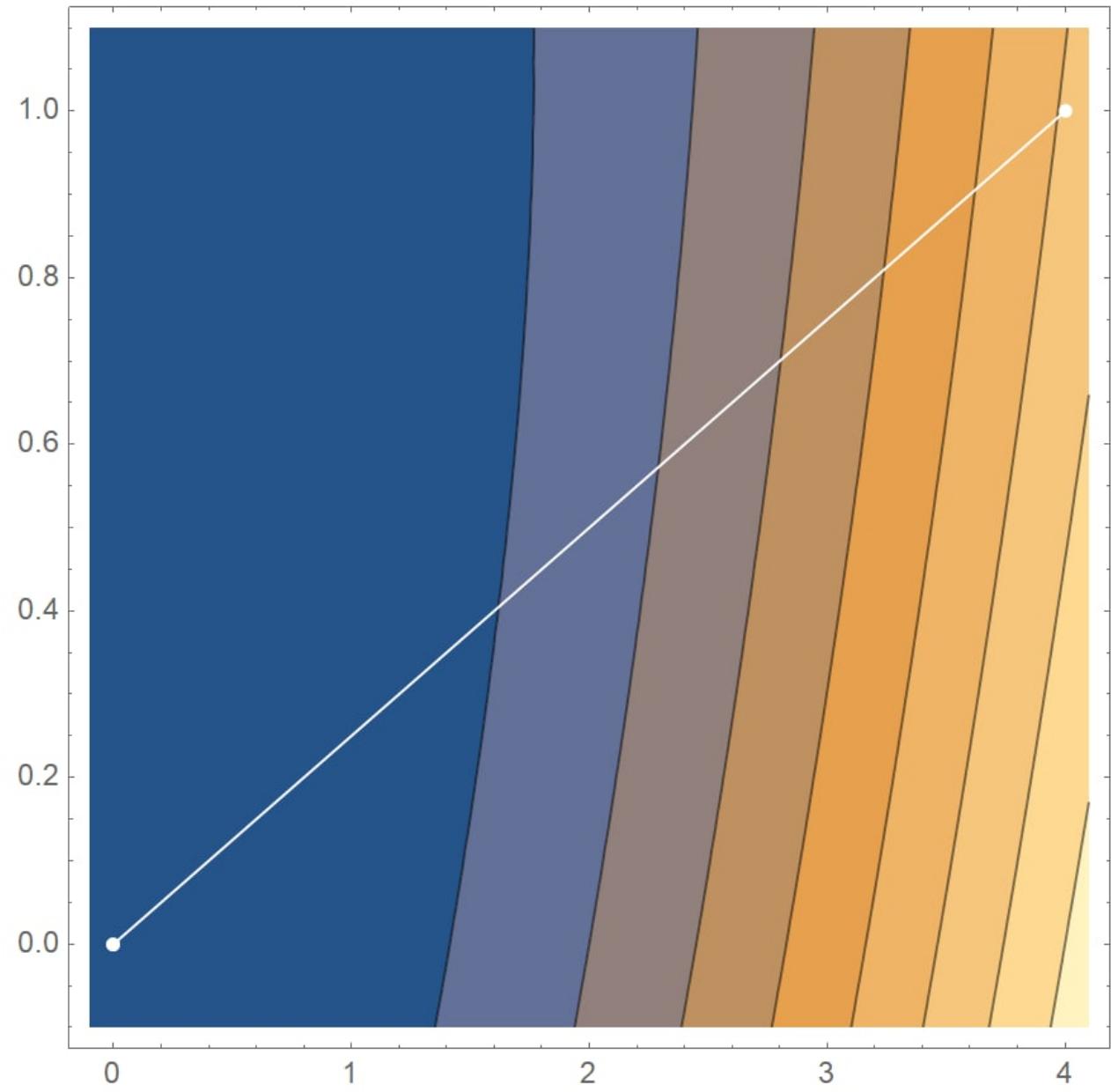
$$f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2 \text{ где } x_0 = (-1.2, 1)^T$$

Для поиска ньютоновского направления используется метод Гаусса.

### 4.1. Классический метод Ньютона

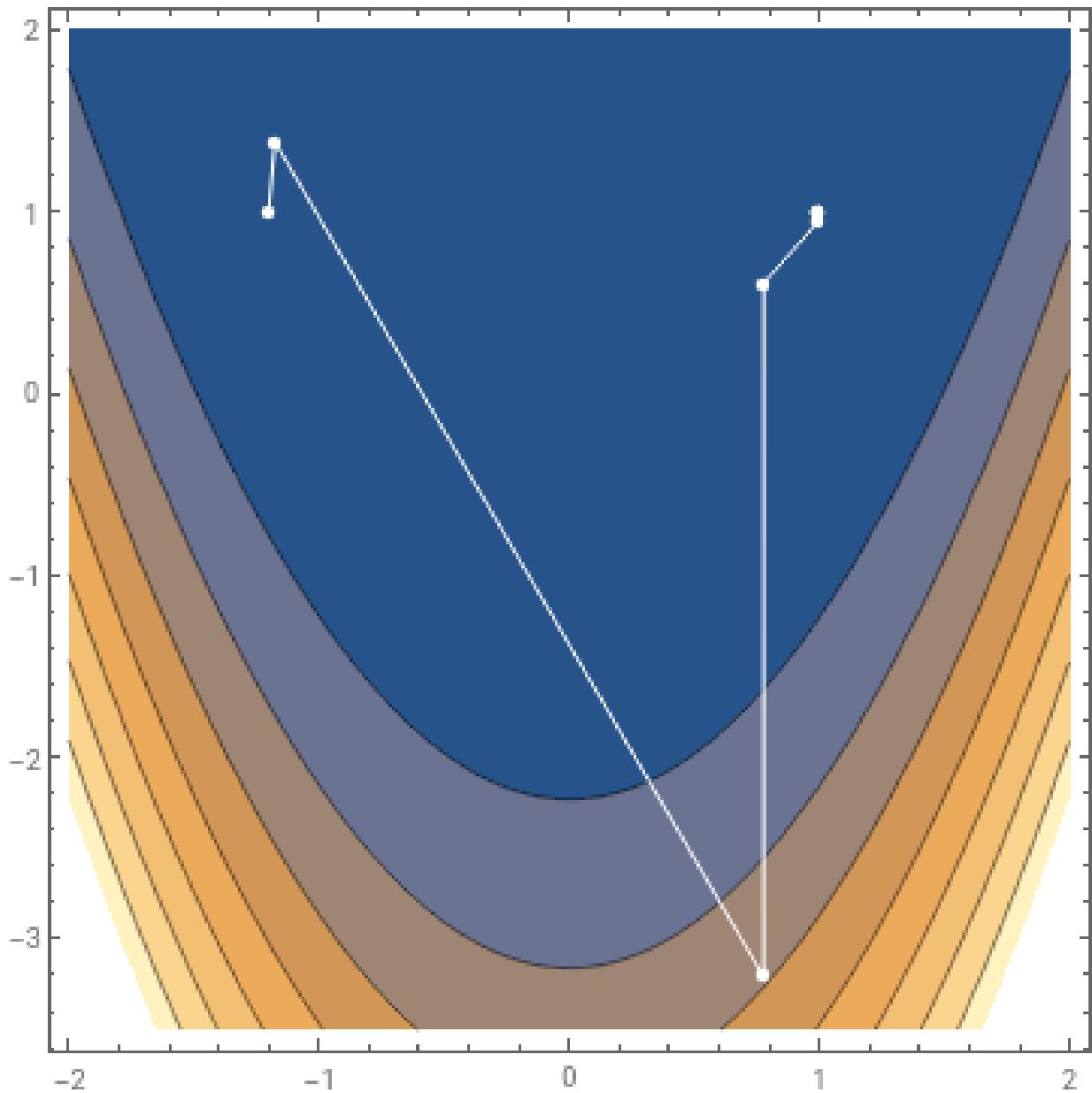
Функция:  $f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2$  где  $x_0 = (4, 1)^T$

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0000000	-0.0000205	-0.0000707	-4.0000205	-1.0000707	0.0000000
1	1.0000000	-0.0000125	-0.0000125	0.0000080	0.0000582	0.0000000
2	1.0000000	-0.0000125	-0.0000125	-0.0000000	-0.0000000	0.0000000



Функция:  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$  где  $x_0 = (-1.2, 1)^T$

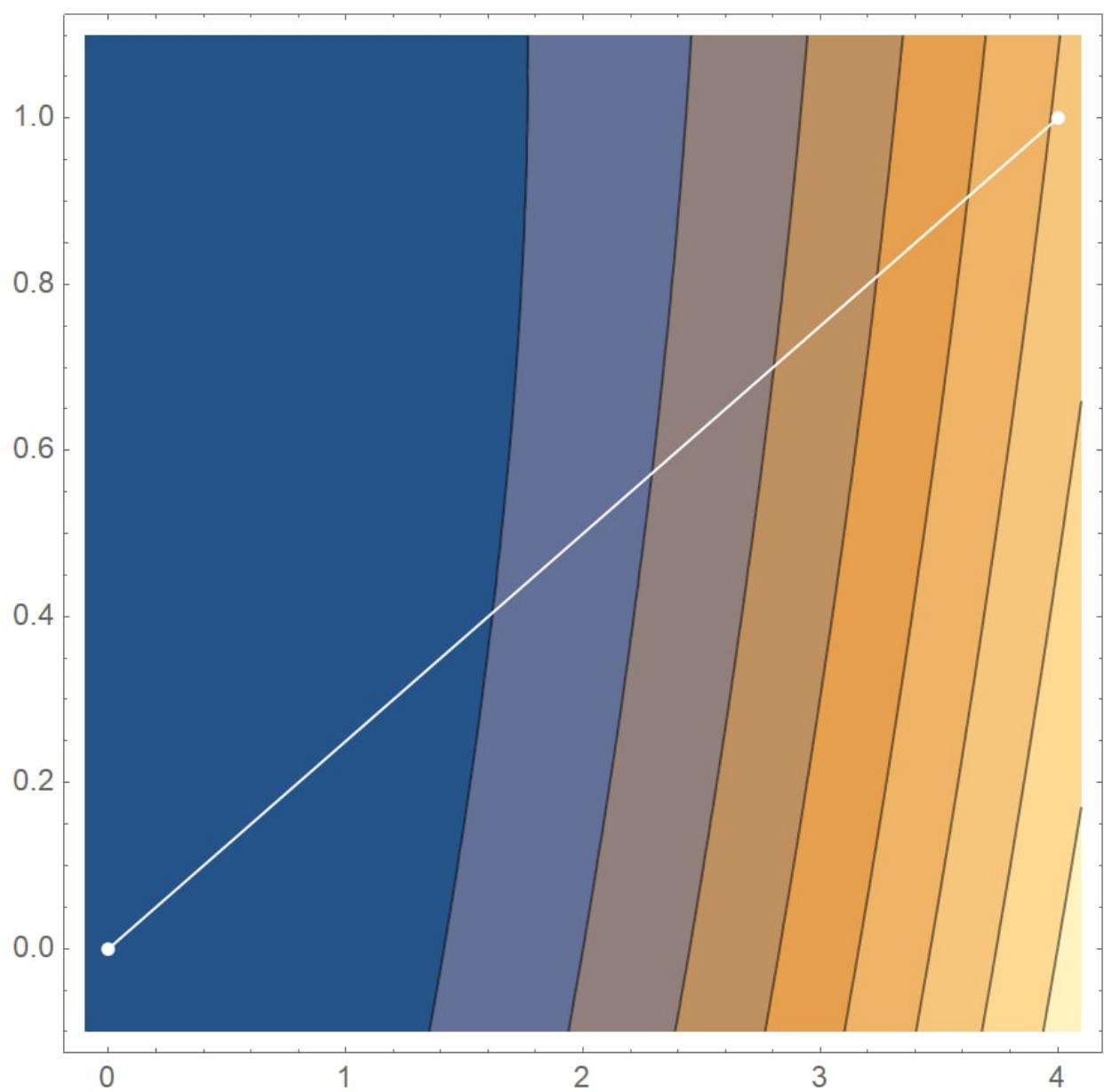
Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0000000	-1.1752973	1.3807088	0.0247027	0.3807088	4.7319562
1	1.0000000	0.7766295	-3.2068503	1.9519268	-4.5875591	1451.6626929
2	1.0000000	0.7769160	0.6035470	0.0002865	3.8103973	0.0497667
3	1.0000000	0.9944222	0.9415638	0.2175063	0.3380168	0.2238718
4	1.0000000	0.9946706	0.9893646	0.0002484	0.0478008	0.0000284
5	1.0000000	0.9970007	0.9939999	0.0023301	0.0046354	0.0000090
6	1.0000000	0.9970099	0.9940237	0.0000092	0.0000237	0.0000089
7	1.0000000	0.9970099	0.9940238	0.0000000	0.0000001	0.0000089



#### 4.2. Метод Ньютона с одномерной оптимизацией методом золотого сечения

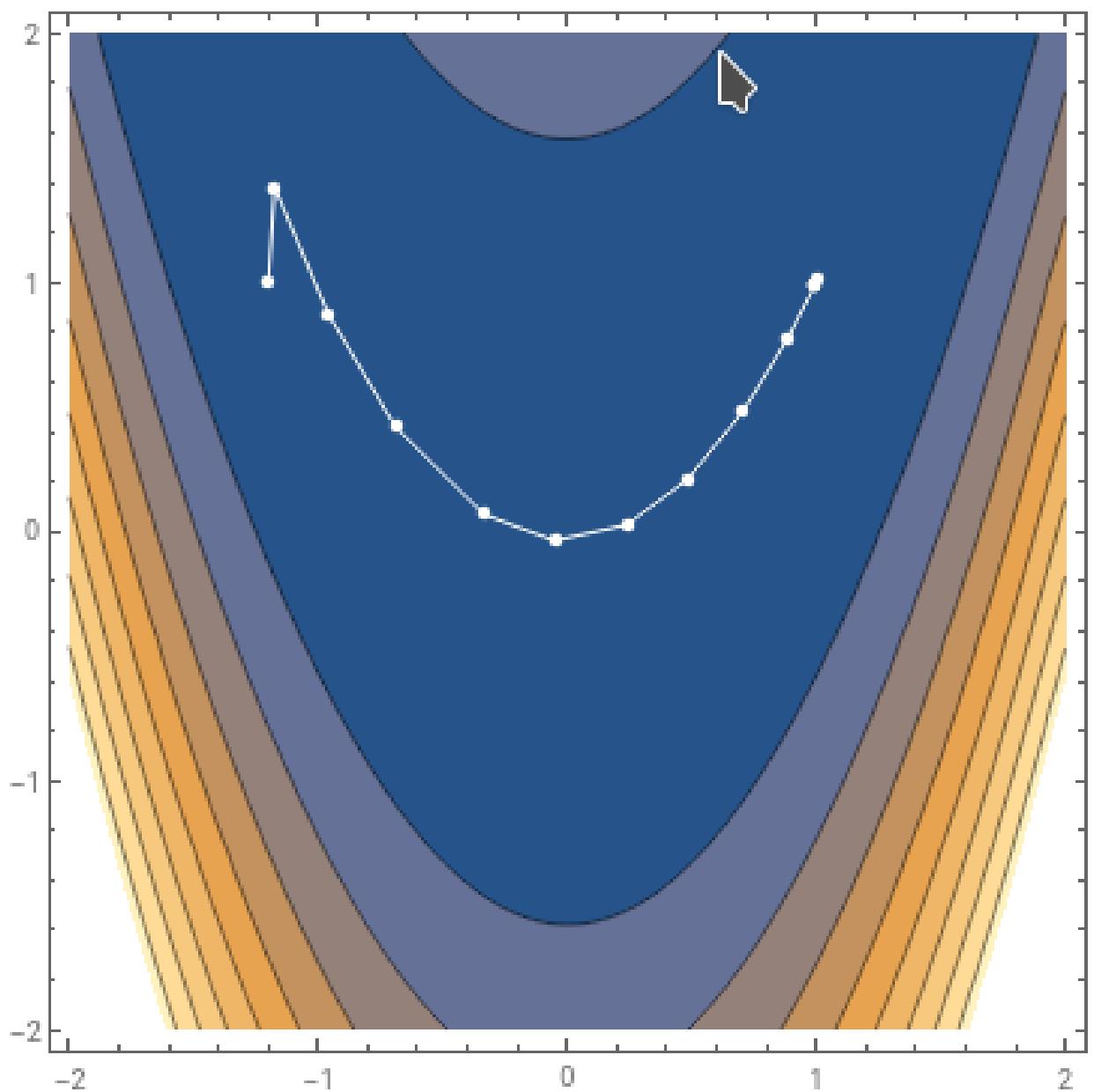
Функция:  $f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2$  где  $x_0 = (4, 1)^T$

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999985	-0.0000144	-0.0000692	-4.0000205	-1.0000707	0.0000000
1	1.0949930	-0.0000123	-0.0000071	0.0000019	0.0000567	0.0000000
2	0.0000422	-0.0000123	-0.0000071	-0.0000002	-0.0000054	0.0000000



Функция:  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$  где  $x_0 = (-1.2, 1)^T$

$k$	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0041967	-1.1751936	1.3823065	0.0247027	0.3807088	4.7316178
1	0.0739520	-0.9595228	0.8753076	2.9163608	-6.8557792	4.0456309
2	1.3861153	-0.6898562	0.4206978	0.1945484	-0.3279740	3.1603598
3	2.5344459	-0.3340534	0.0696941	0.1403868	-0.1384933	1.9552388
4	1.9895399	-0.0509768	-0.0360816	0.1422824	-0.0531659	1.2541679
5	2.4084221	0.2387900	0.0275248	0.1203139	0.0264100	0.6664414
6	2.2073311	0.4821507	0.2088475	0.1102511	0.0821457	0.3239667
7	2.4988044	0.7076356	0.4852989	0.0902371	0.1106335	0.1093450
8	2.5026798	0.8852453	0.7753187	0.0709678	0.1158837	0.0201252
9	2.8402177	1.0044694	1.0100798	0.0419771	0.0826560	0.0001457
10	0.7603725	0.9971968	0.9946133	-0.0095645	-0.0203407	0.0000123
11	0.8540904	0.9970307	0.9940968	-0.0001945	-0.0006048	0.0000089
12	0.2229746	0.9970260	0.9940805	-0.0000208	-0.0000731	0.0000089
13	0.0016309	0.9970260	0.9940804	-0.0000161	-0.0000567	0.0000089

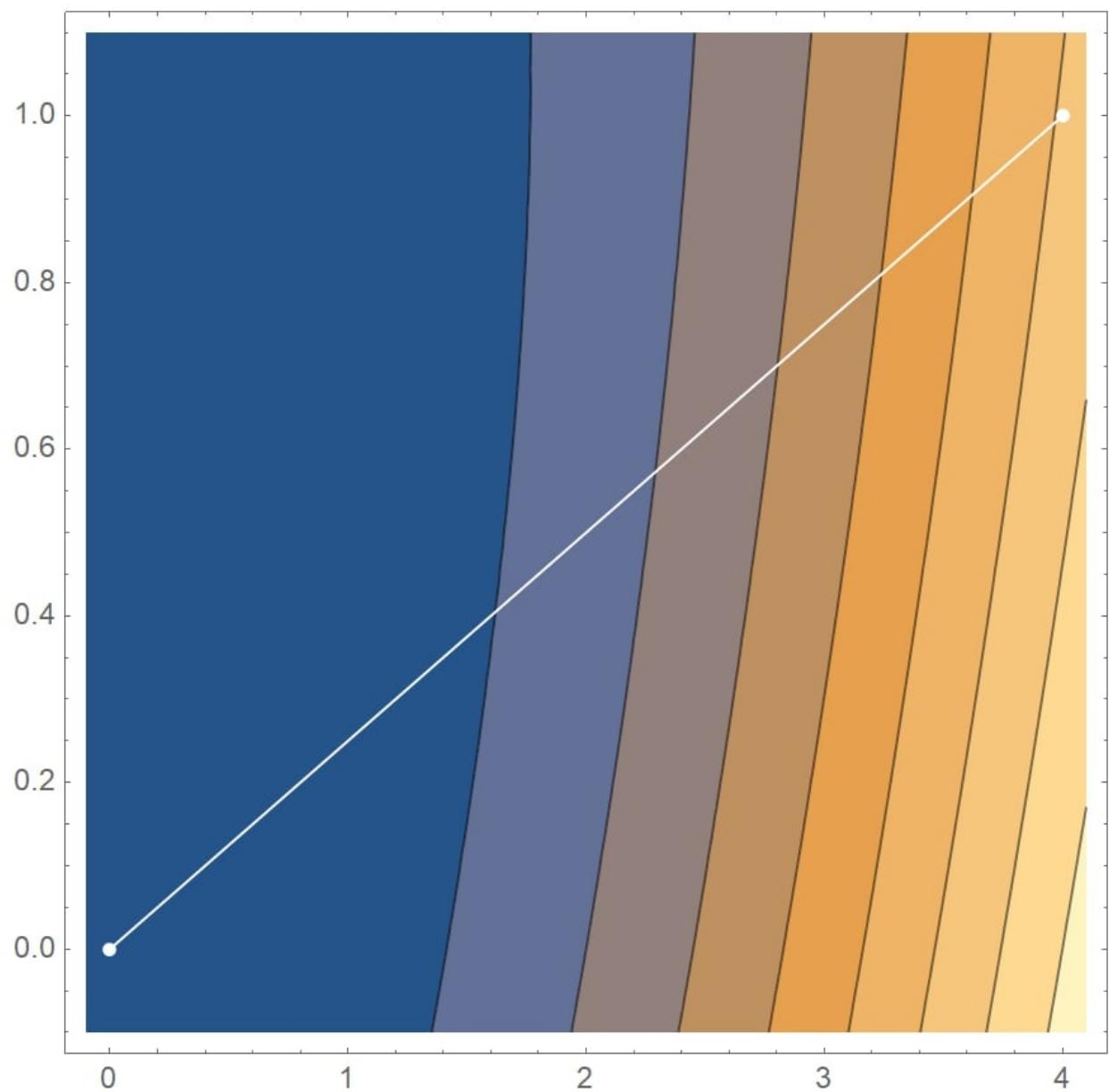


### 4.3. Метод Ньютона с направлением спуска

В качестве одномерной оптимизации также выбран метод золотого сечения.

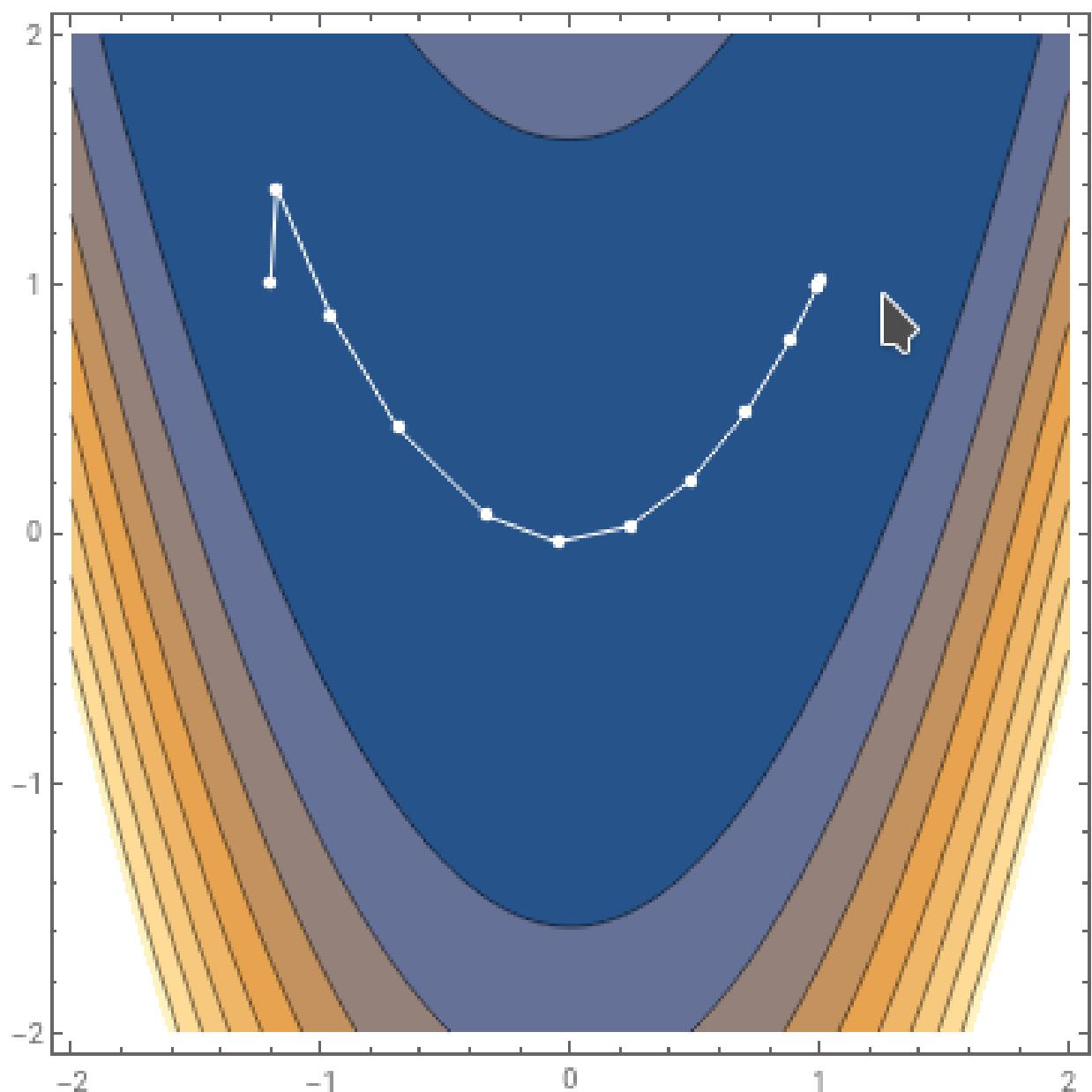
Функция:  $f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2$  где  $x_0 = (4, 1)^T$

Итерация	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	0.9999985	-0.0000144	-0.0000692	-4.0000205	-1.0000707	0.0000000
1	1.0949930	-0.0000123	-0.0000071	0.0000019	0.0000567	0.0000000
2	0.0000422	-0.0000123	-0.0000071	-0.0000002	-0.0000054	0.0000000



Функция:  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$  где  $x_0 = (-1.2, 1)^T$

$k$	$\alpha_k$	$x_{k_1}$	$x_{k_2}$	$p_{k_1}$	$p_{k_2}$	$f(x_k)$
0	1.0041967	-1.1751936	1.3823065	0.0247027	0.3807088	4.7316178
1	0.0739520	-0.9595228	0.8753076	2.9163608	-6.8557792	4.0456309
2	1.3861153	-0.6898562	0.4206978	0.1945484	-0.3279740	3.1603598
3	2.5344459	-0.3340534	0.0696941	0.1403868	-0.1384933	1.9552388
4	1.9895399	-0.0509768	-0.0360816	0.1422824	-0.0531659	1.2541679
5	2.4084221	0.2387900	0.0275248	0.1203139	0.0264100	0.6664414
6	2.2073311	0.4821507	0.2088475	0.1102511	0.0821457	0.3239667
7	2.4988044	0.7076356	0.4852989	0.0902371	0.1106335	0.1093450
8	2.5026798	0.8852453	0.7753187	0.0709678	0.1158837	0.0201252
9	2.8402177	1.0044694	1.0100798	0.0419771	0.0826560	0.0001457
10	0.7603725	0.9971968	0.9946133	-0.0095645	-0.0203407	0.0000123
11	0.8540904	0.9970307	0.9940968	-0.0001945	-0.0006048	0.0000089
12	0.2229746	0.9970260	0.9940805	-0.0000208	-0.0000731	0.0000089
13	0.0016309	0.9970260	0.9940804	-0.0000161	-0.0000567	0.0000089



#### 4.4. Сравнение количества итераций

Функция:  $f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2$  где  $x_0 = (4, 1)^T$

Классический	3
С одномерной оптимизацией	3
С направлением спуска	3
Наискорейший спуск	23

Функция:  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$  где  $x_0 = (-1.2, 1)^T$

Классический	8
С одномерной оптимизацией	13
С направлением спуска	13
Наискорейший спуск	89

#### 4.5. Выводы по исследованиям на заданных функциях

- $f(x) = x_1^2 + x_2^2 - 1.2 \cdot x_1 \cdot x_2$  является квадратичной, поэтому все методы сходятся с достаточно высокой скоростью.  
В ходе исследования классический метод Ньютона показал наилучшие результаты.

## 5. Квазиньютоновские методы

Проведем сравнение методов Давидона-Флетчера-Пауэлла, Пауэла и классического метода Ньютона на четырех функциях с различным начальным приближением:

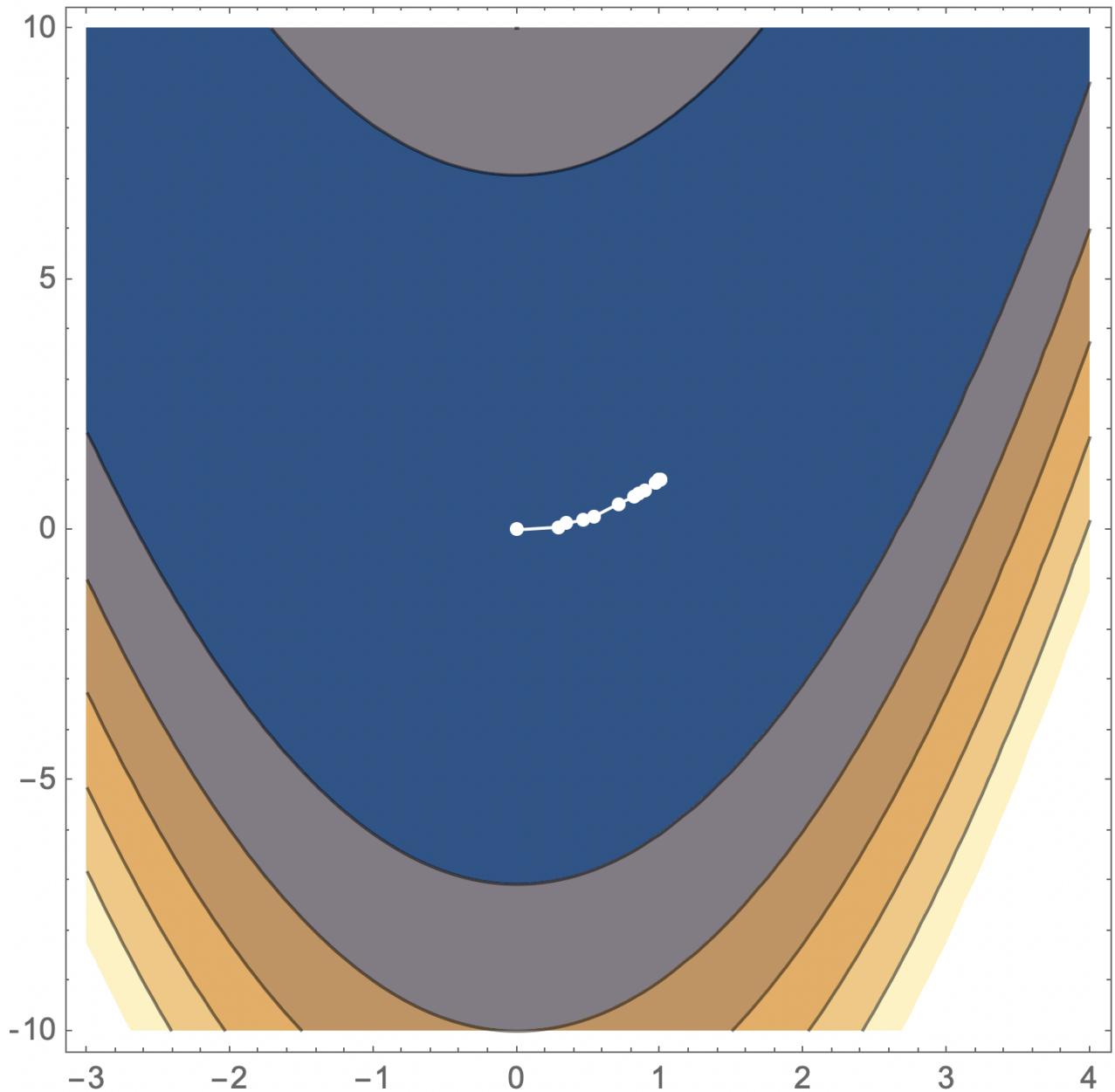
- $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$
- $f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$
- $f(x) = (x_1 + 10 \cdot x_2)^2 + 5 \cdot (x_3 - x_4)^2 + (x_2 - 2 \cdot x_3)^4 + 10 \cdot (x_1 - x_4)^4$
- $f(x) = 100 - \frac{2}{1+(\frac{x_1-1}{2})^2+(\frac{x_2-1}{3})^2} - \frac{1}{1+(\frac{x_1-2}{2})^2+(\frac{x_2-1}{3})^2}$

## 5.1. Метод Давидона-Флетчера-Пауэлла

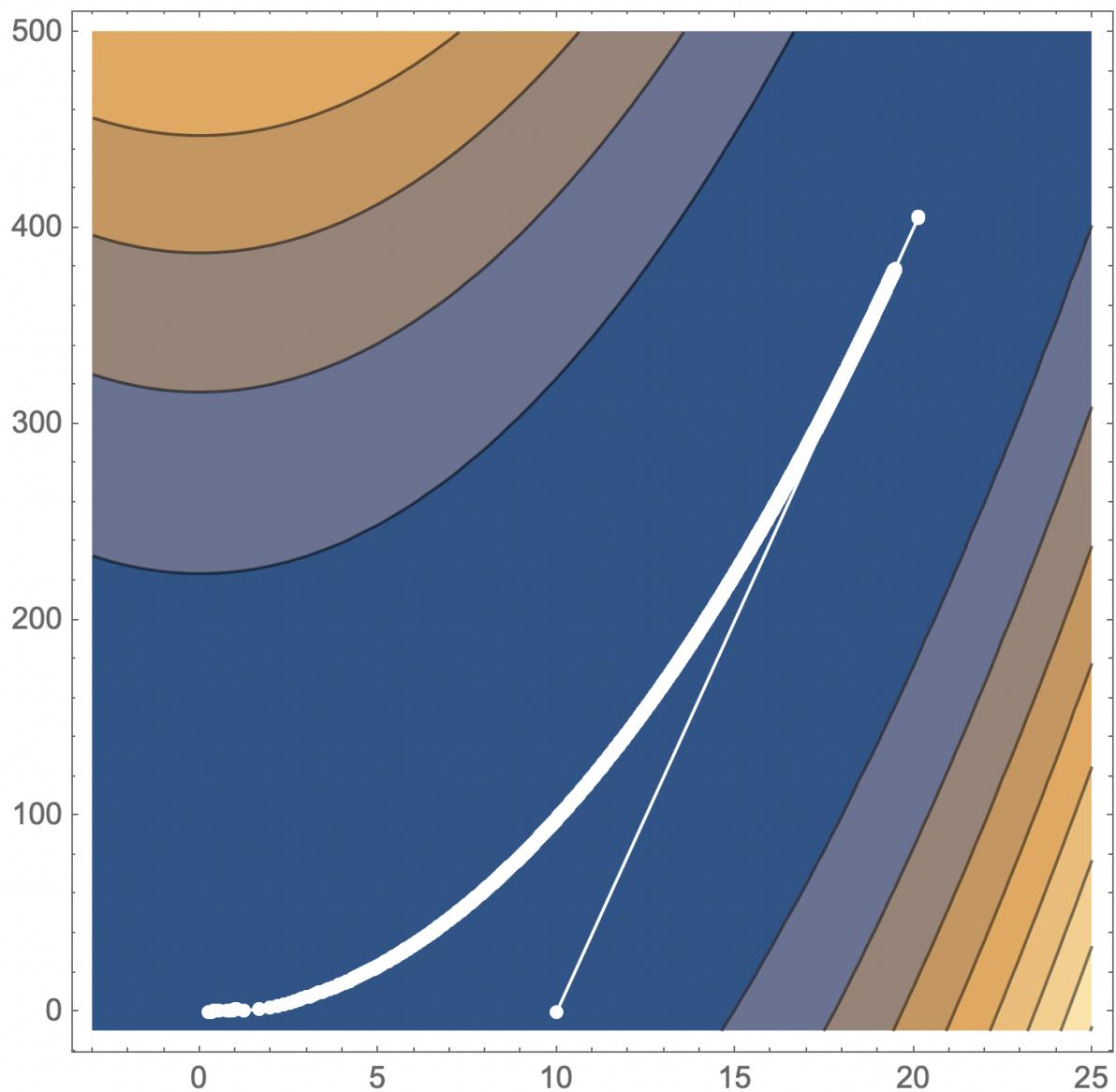
5.1.1.  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$

Начальное приближение:  $(x, y) = (0.0, 0.0)$

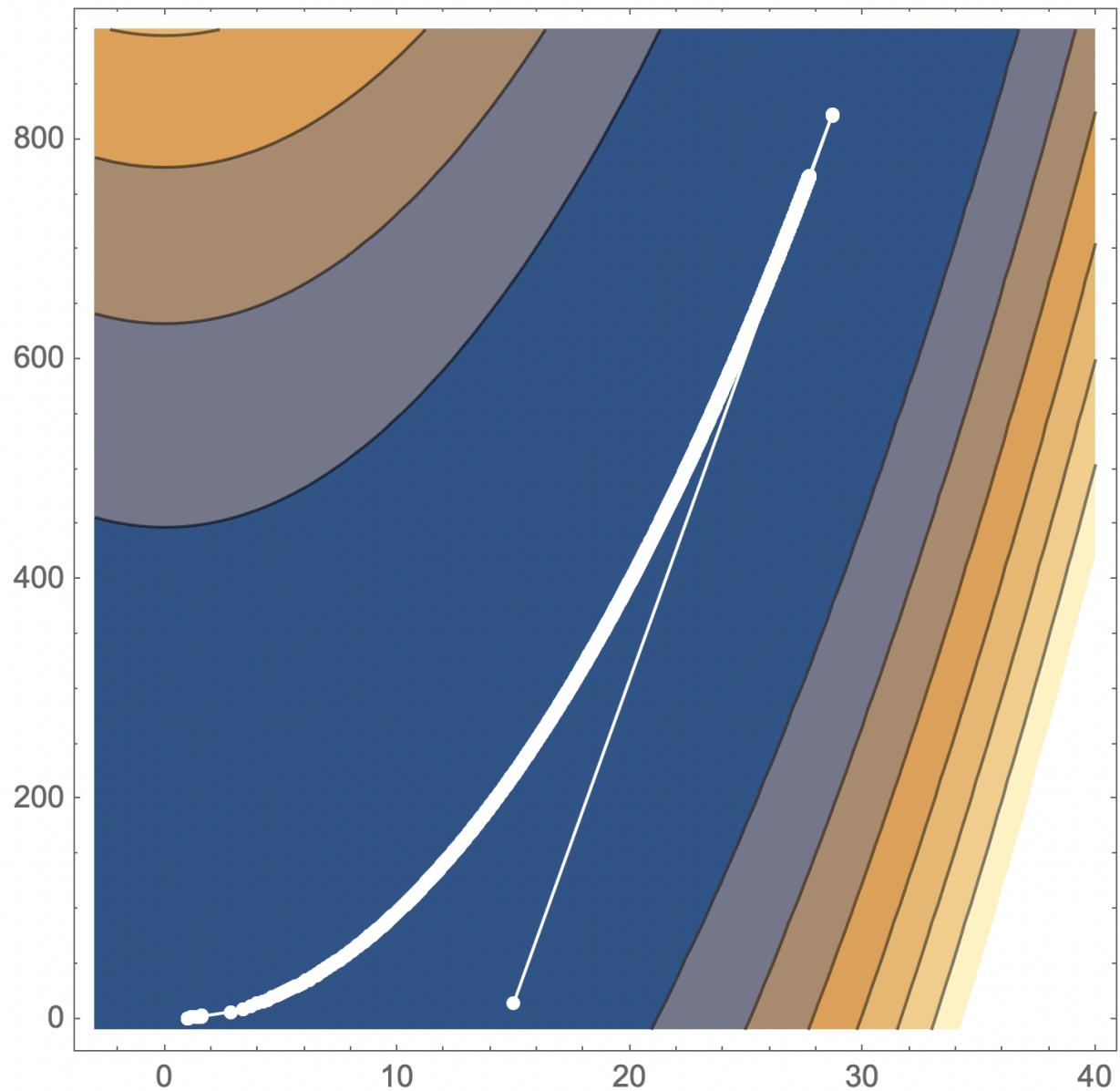
Границы одномерного поиска: -10 до 15.



Начальное приближение:  $(x, y) = (10.0, 0.0)$   
Границы одномерного поиска: -10 до 15.

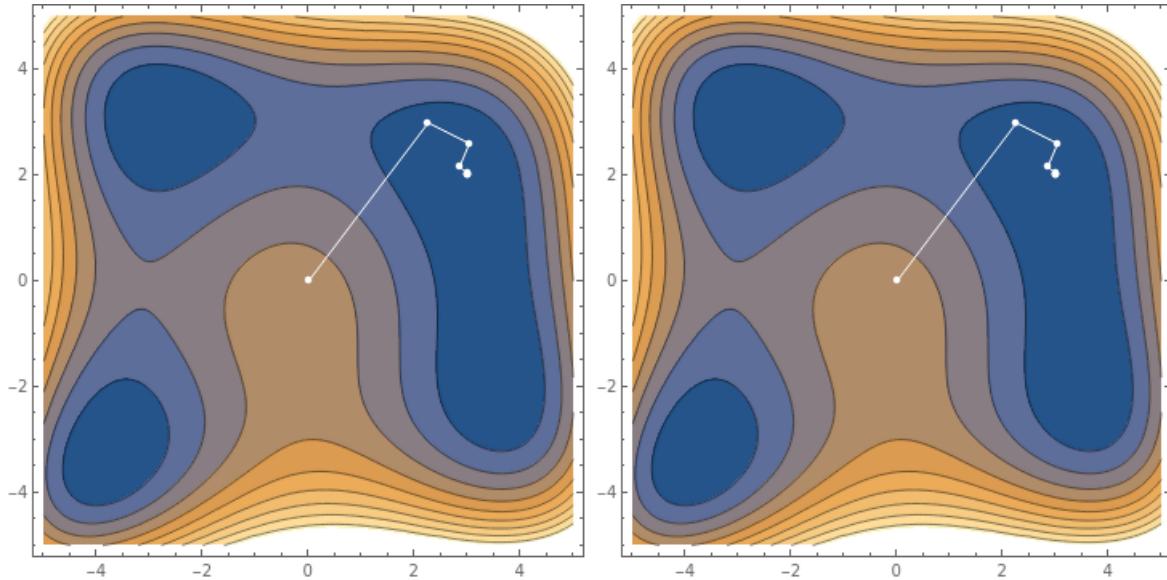


Начальное приближение:  $(x, y) = (15.0, 15.0)$   
Границы одномерного поиска: -10 до 15.



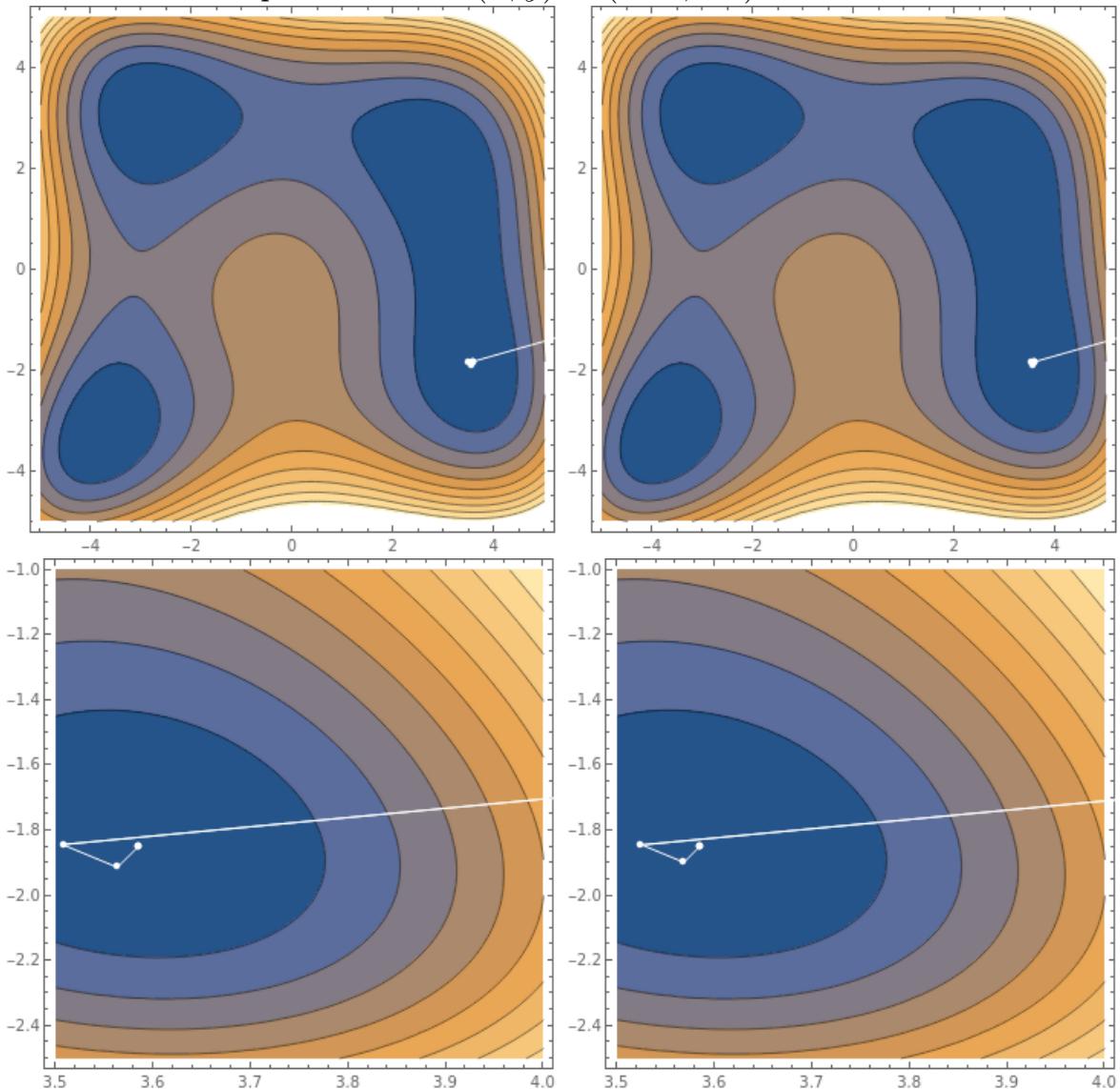
$$5.1.2. \ f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

Начальное приближение:  $(x, y) = (0.0, 0.0)$



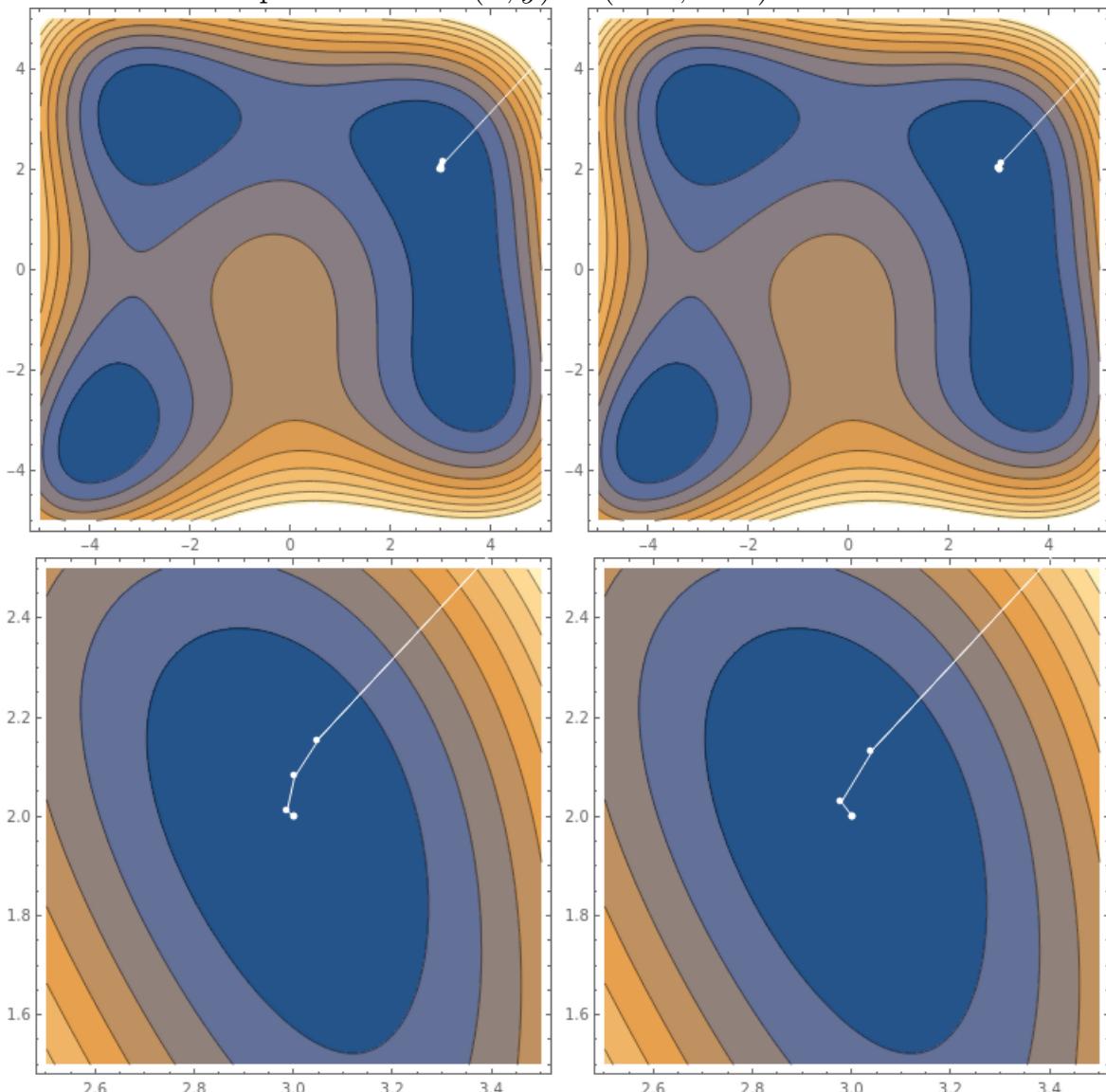
На левой картинке используются границы одномерного поиска от  $-10$  до  $10$ , а на правой от  $-1000$  до  $1000$ .

Начальное приближение:  $(x, y) = (10.0, 0.0)$



На левой картинке используются границы одномерного поиска от  $-10$  до  $10$ , а на правой от  $-1000$  до  $1000$ .

Начальное приближение:  $(x, y) = (15.0, 15.0)$

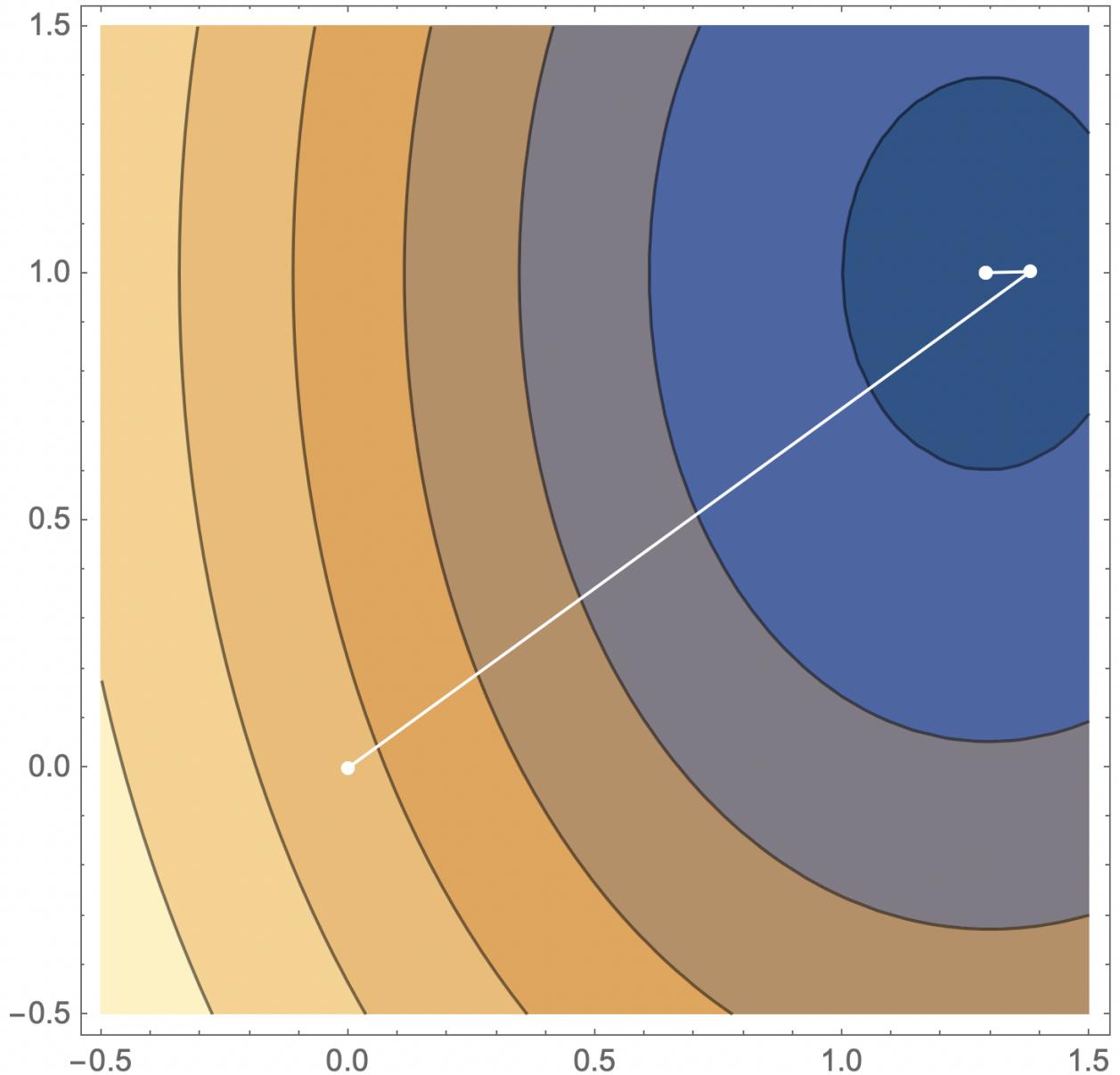


На левой картинке используются границы одномерного поиска от  $-10$  до  $10$ , а на правой от  $-1000$  до  $1000$ .

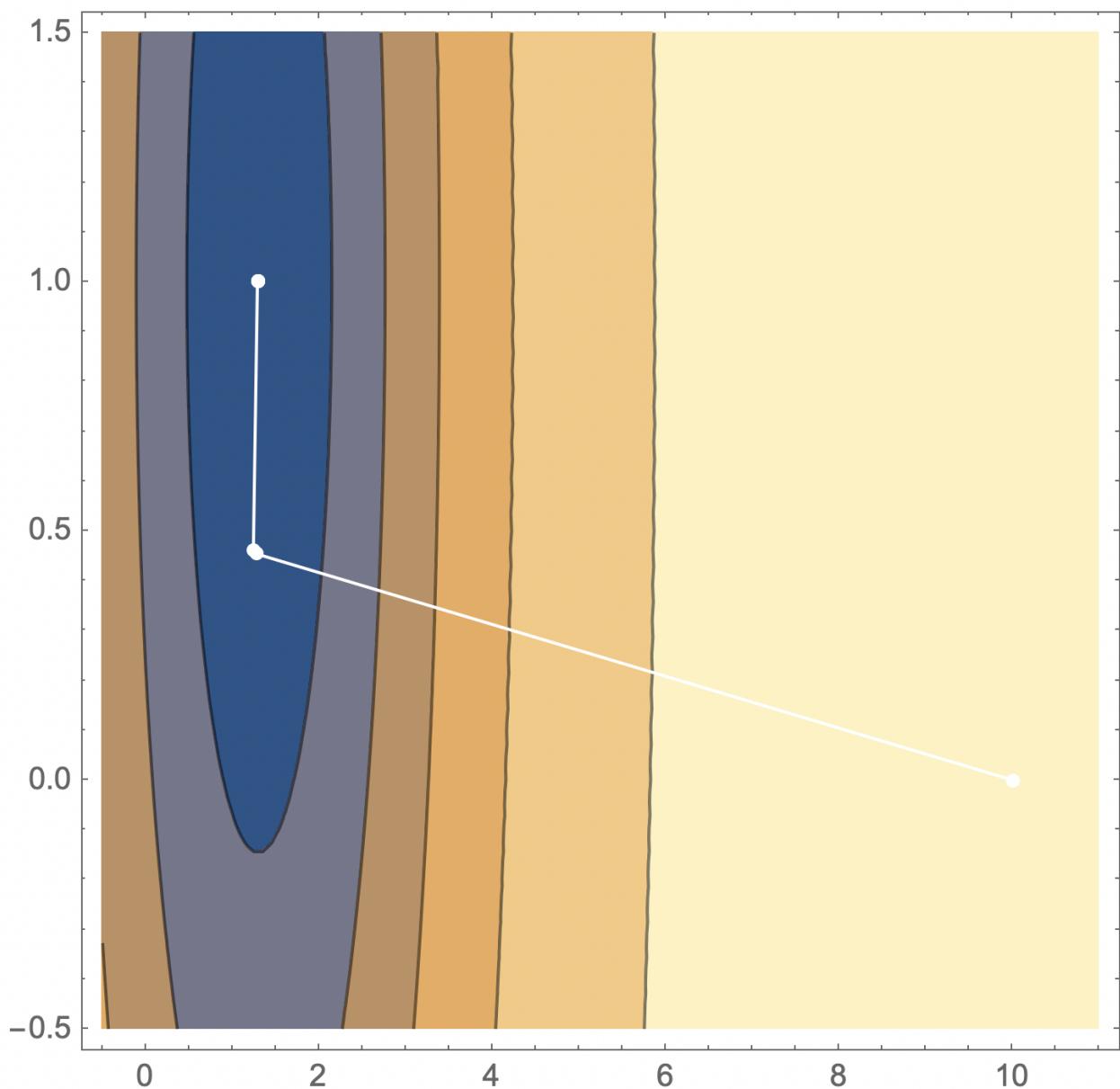
$$5.1.3. \quad f(x) = 100 - \frac{2}{1+(\frac{x_1-1}{2})^2+(\frac{x_2-1}{3})^2} - \frac{1}{1+(\frac{x_1-2}{2})^2+(\frac{x_2-1}{3})^2}$$

Начальное приближение:  $(x, y) = (0.0, 0.0)$

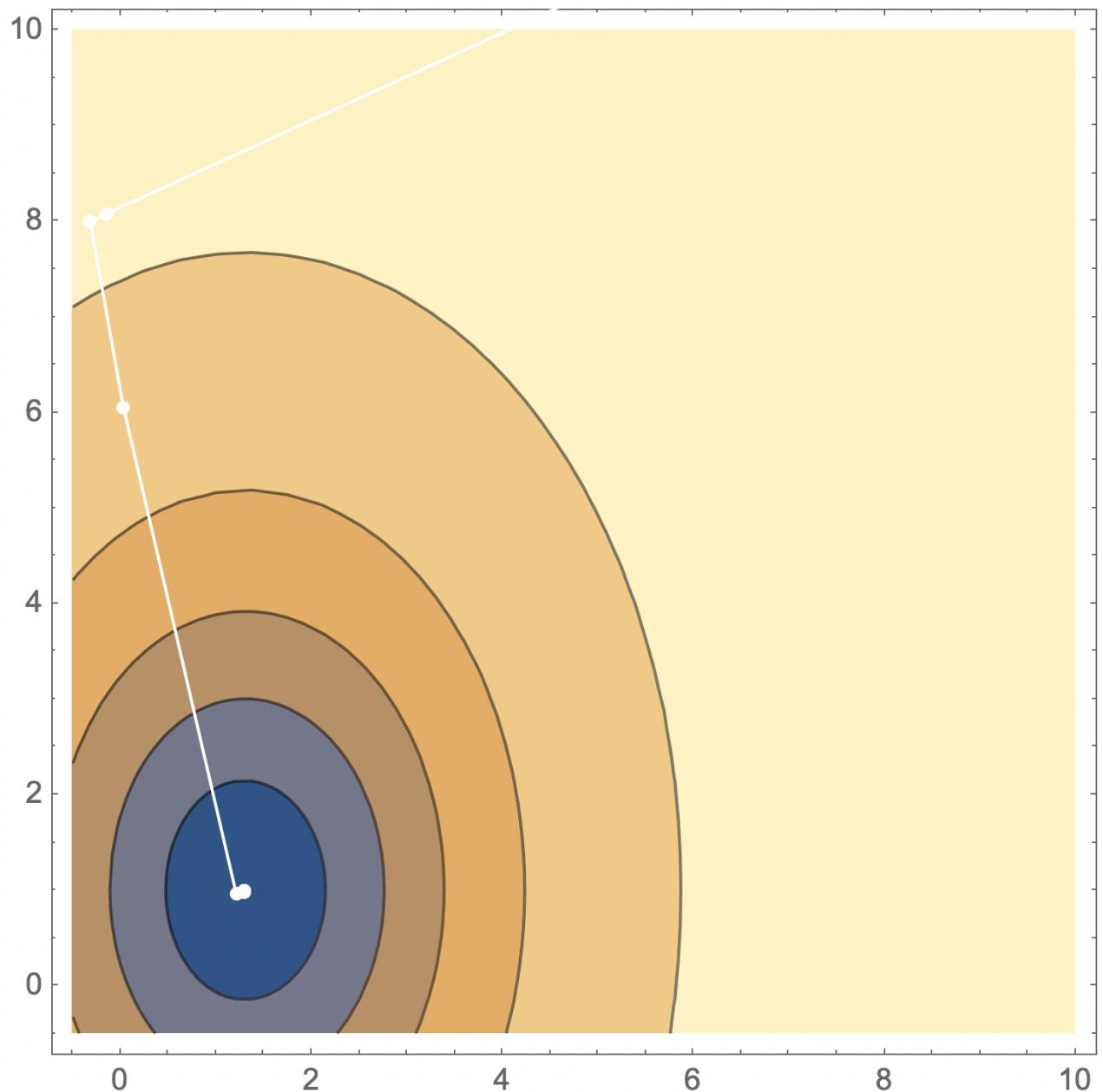
Границы одномерного поиска: -10 до 15.



Начальное приближение:  $(x, y) = (10.0, 0.0)$   
Границы одномерного поиска: -10 до 15.



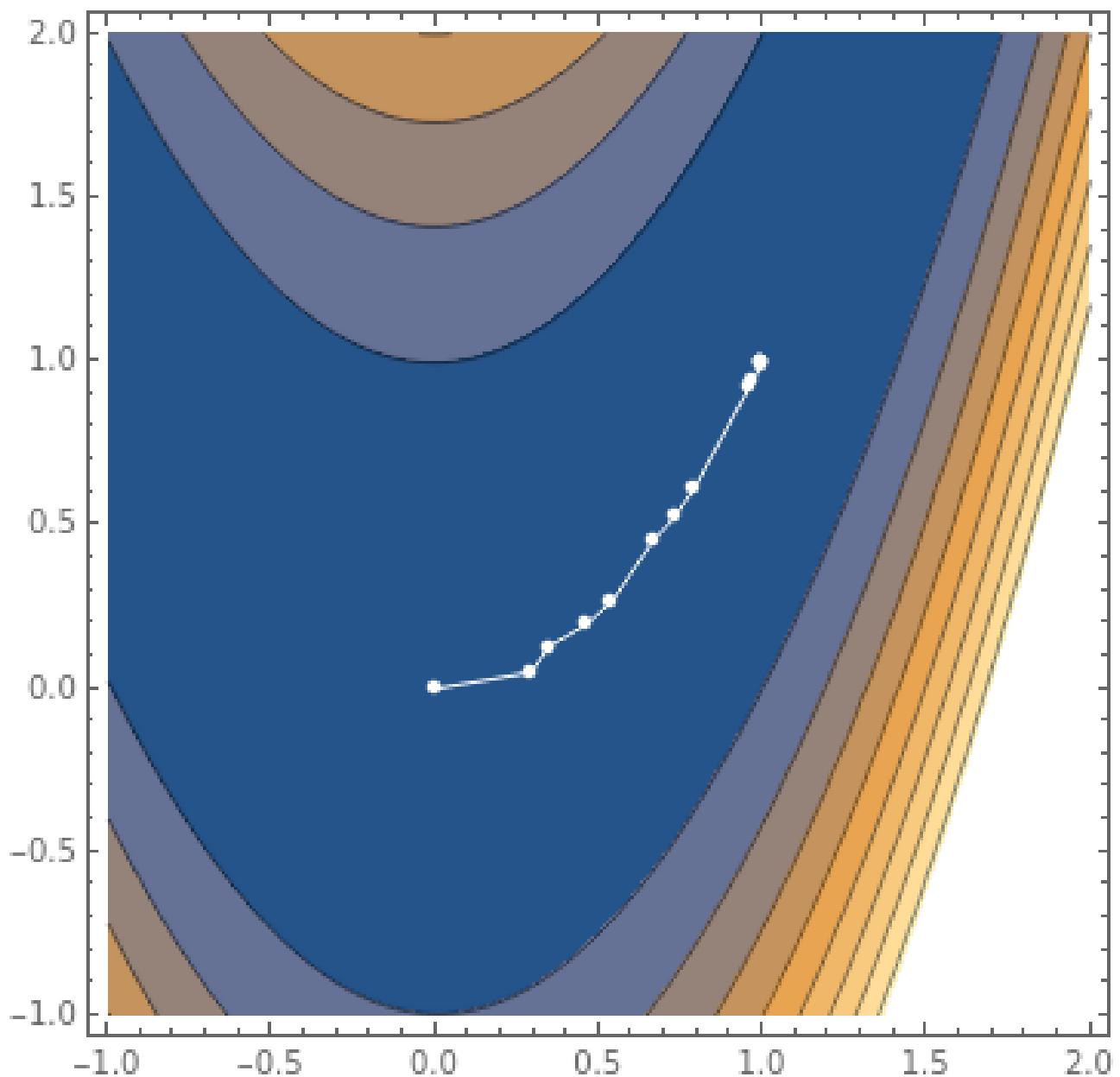
Начальное приближение:  $(x, y) = (15.0, 15.0)$   
Границы одномерного поиска: -10 до 15.



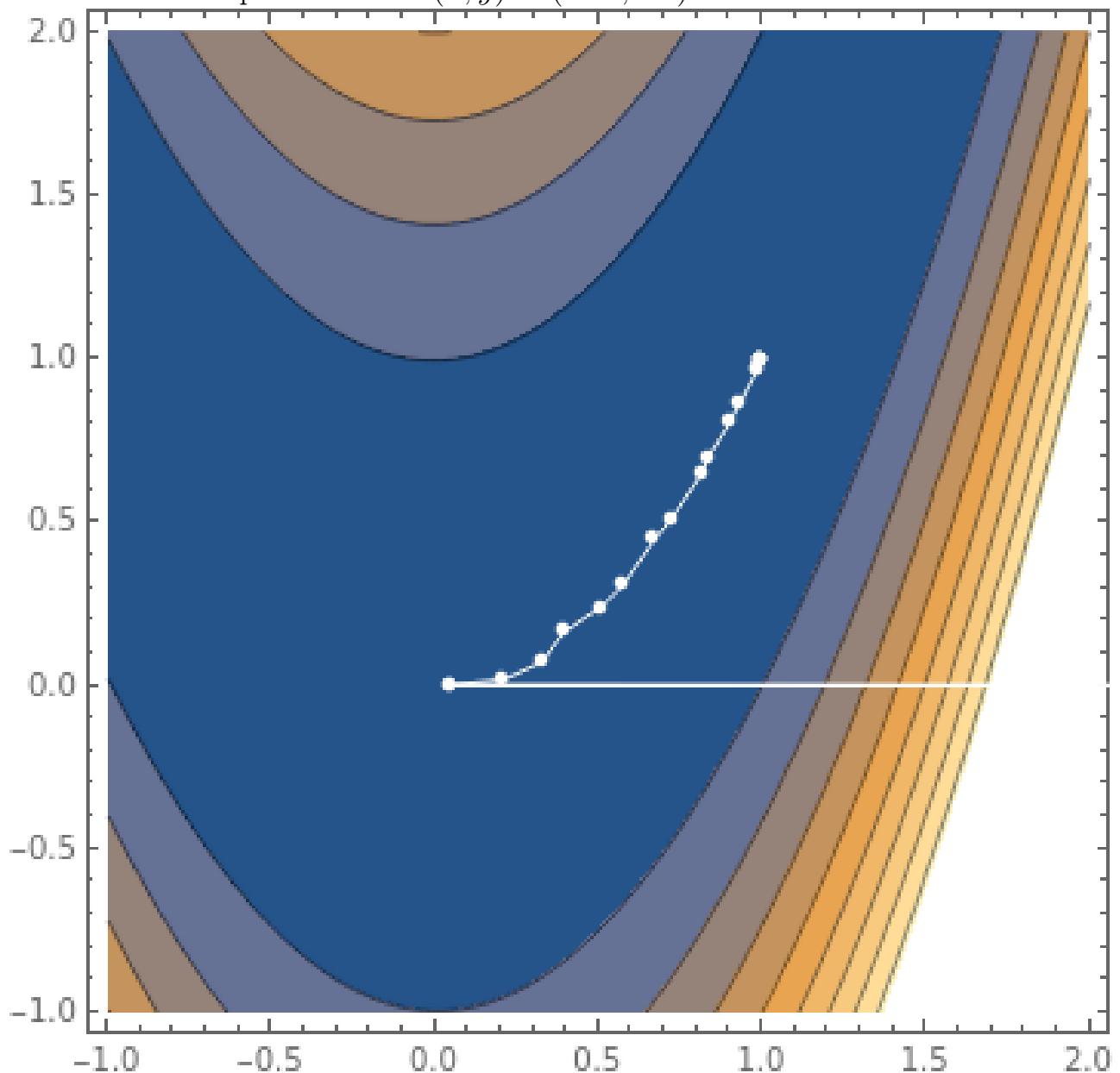
## 5.2. Пауэлла

5.2.1.  $f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$

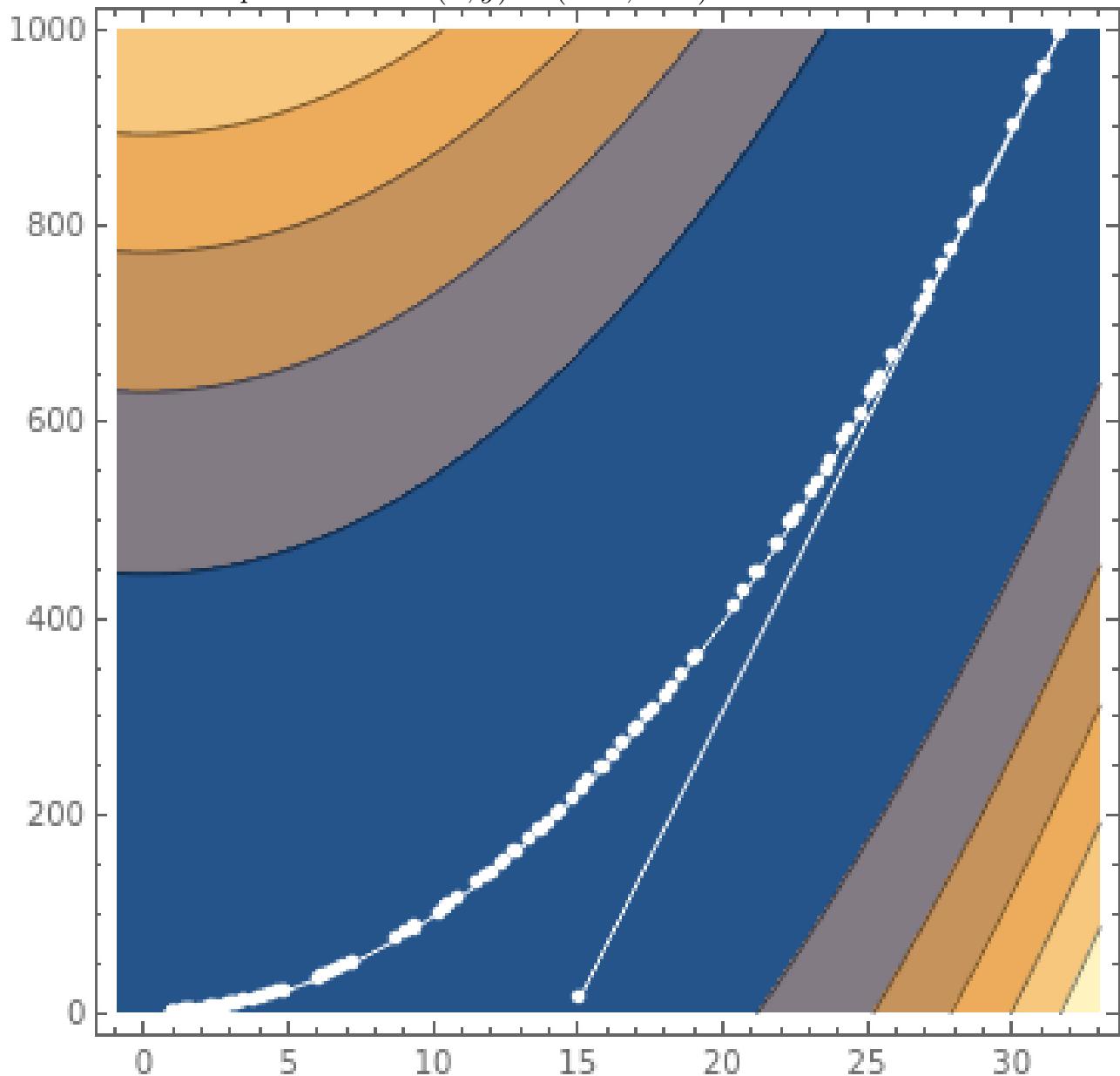
Начальное приближение:  $(x, y) = (0.0, 0.0)$



Начальное приближение:  $(x, y) = (10.0, 0.0)$

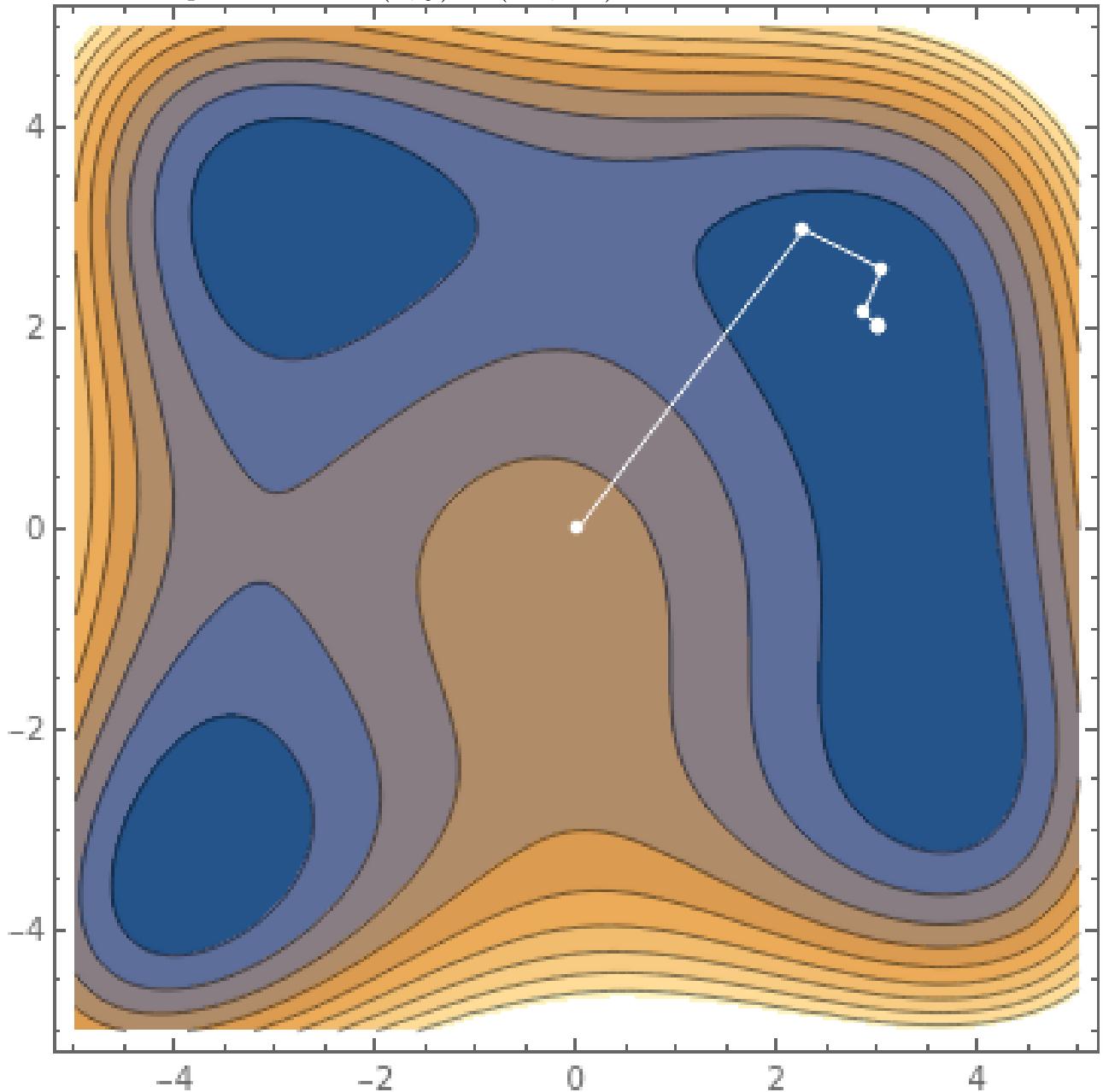


Начальное приближение:  $(x, y) = (15.0, 15.0)$

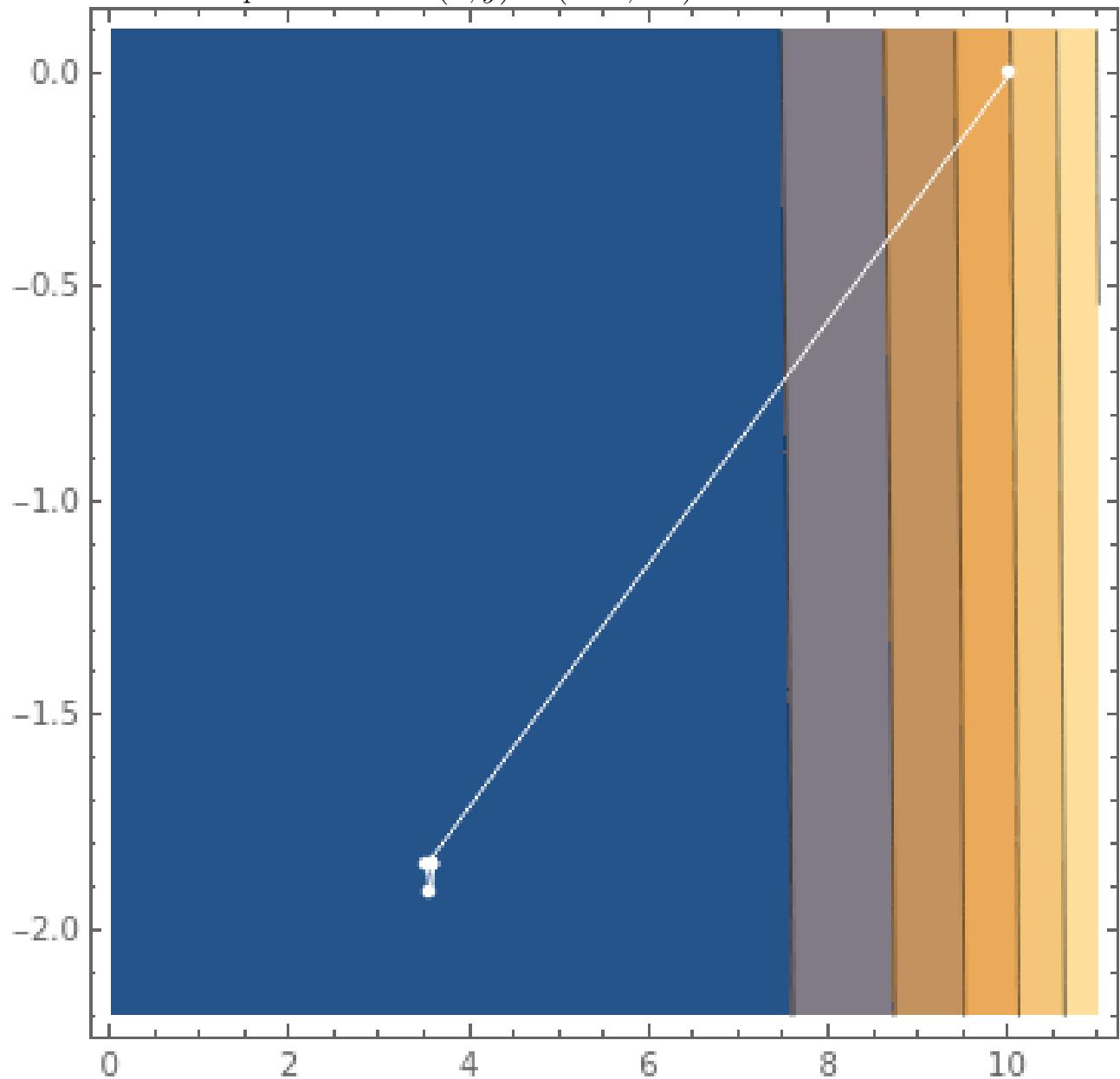


$$5.2.2. \quad f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

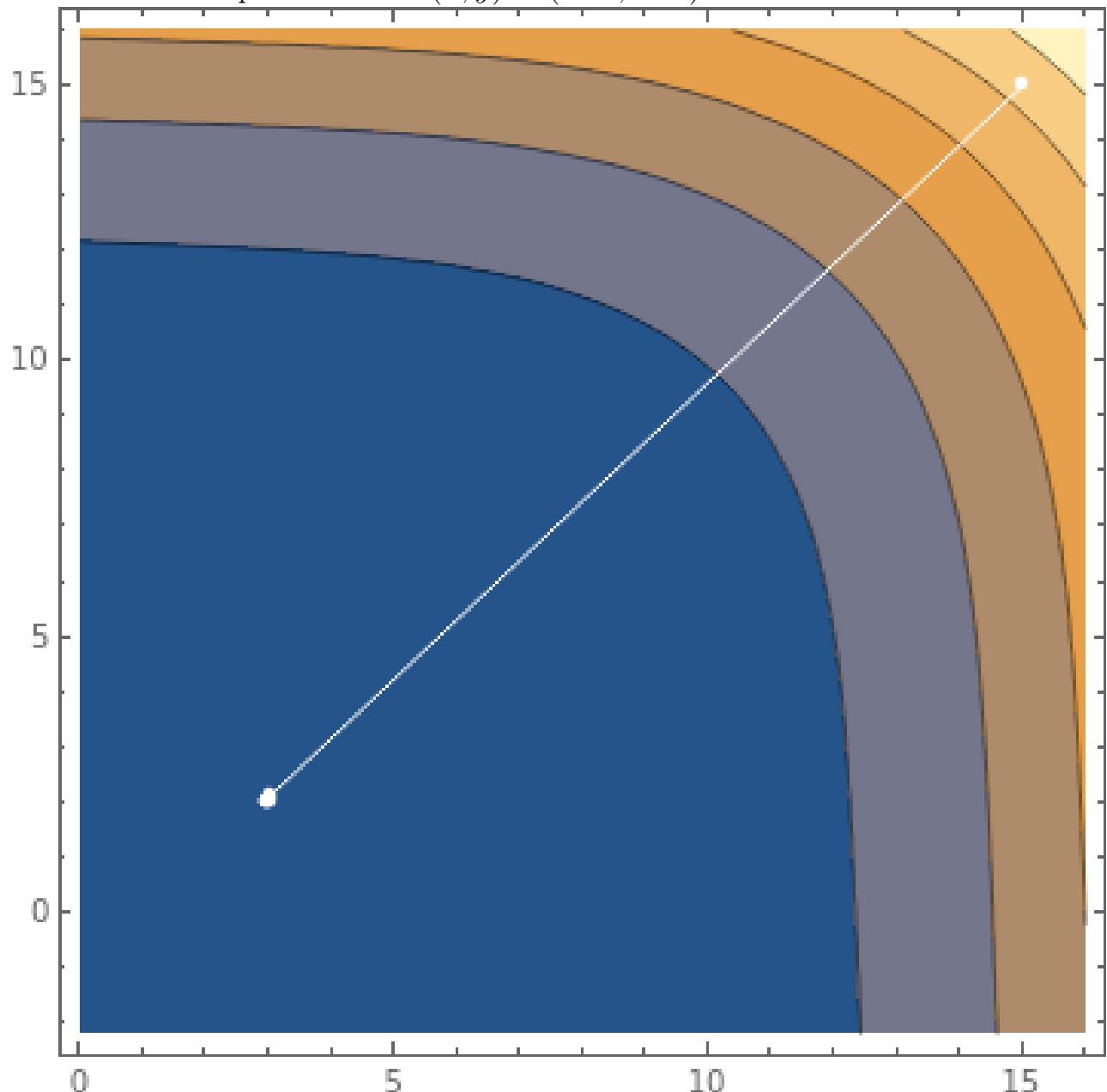
Начальное приближение:  $(x, y) = (0.0, 0.0)$



Начальное приближение:  $(x, y) = (10.0, 0.0)$

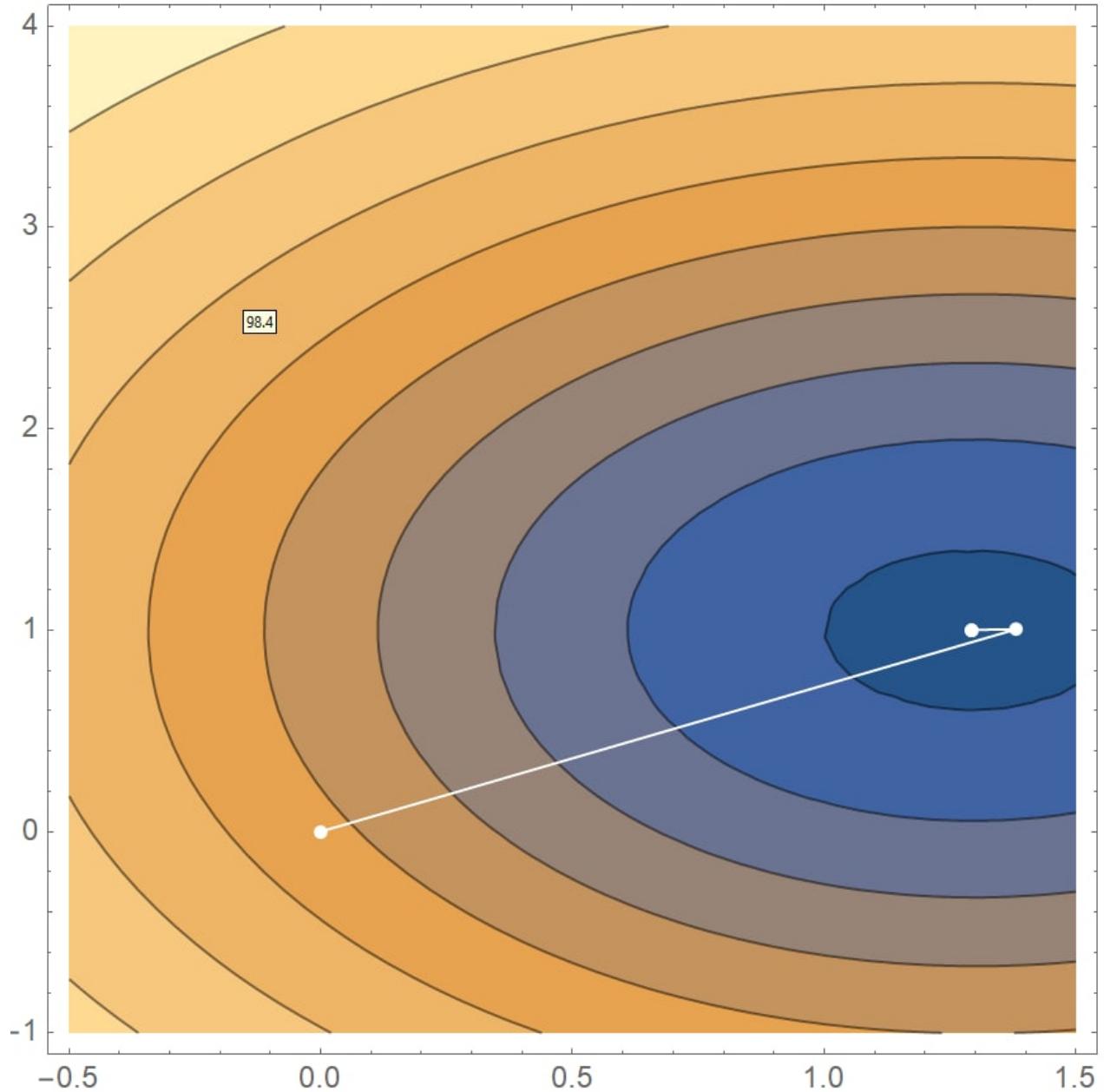


Начальное приближение:  $(x, y) = (15.0, 15.0)$

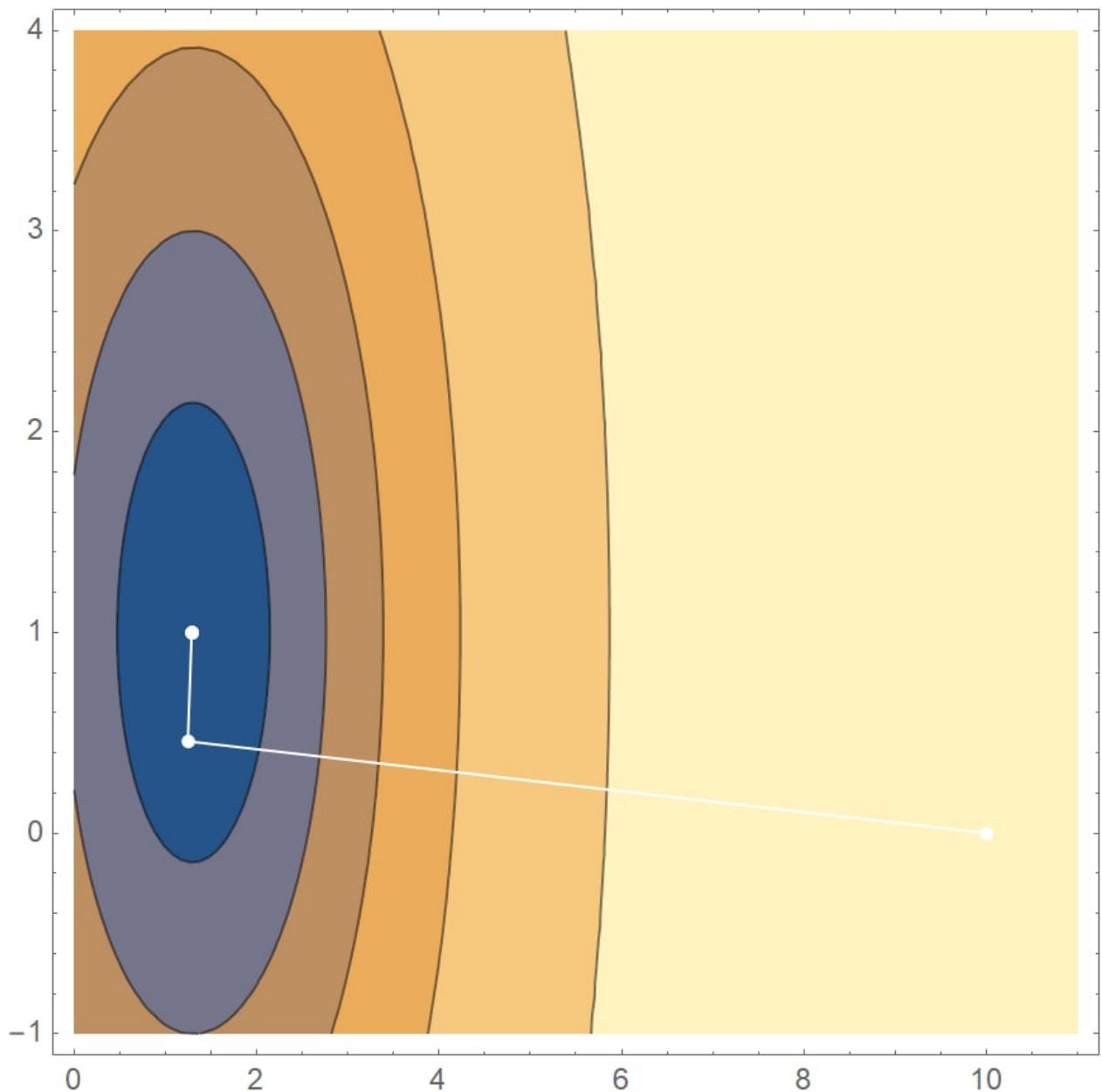


$$5.2.3. \quad f(x) = 100 - \frac{2}{1+(\frac{x_1-1}{2})^2+(\frac{x_2-1}{3})^2} - \frac{1}{1+(\frac{x_1-2}{2})^2+(\frac{x_2-1}{3})^2}$$

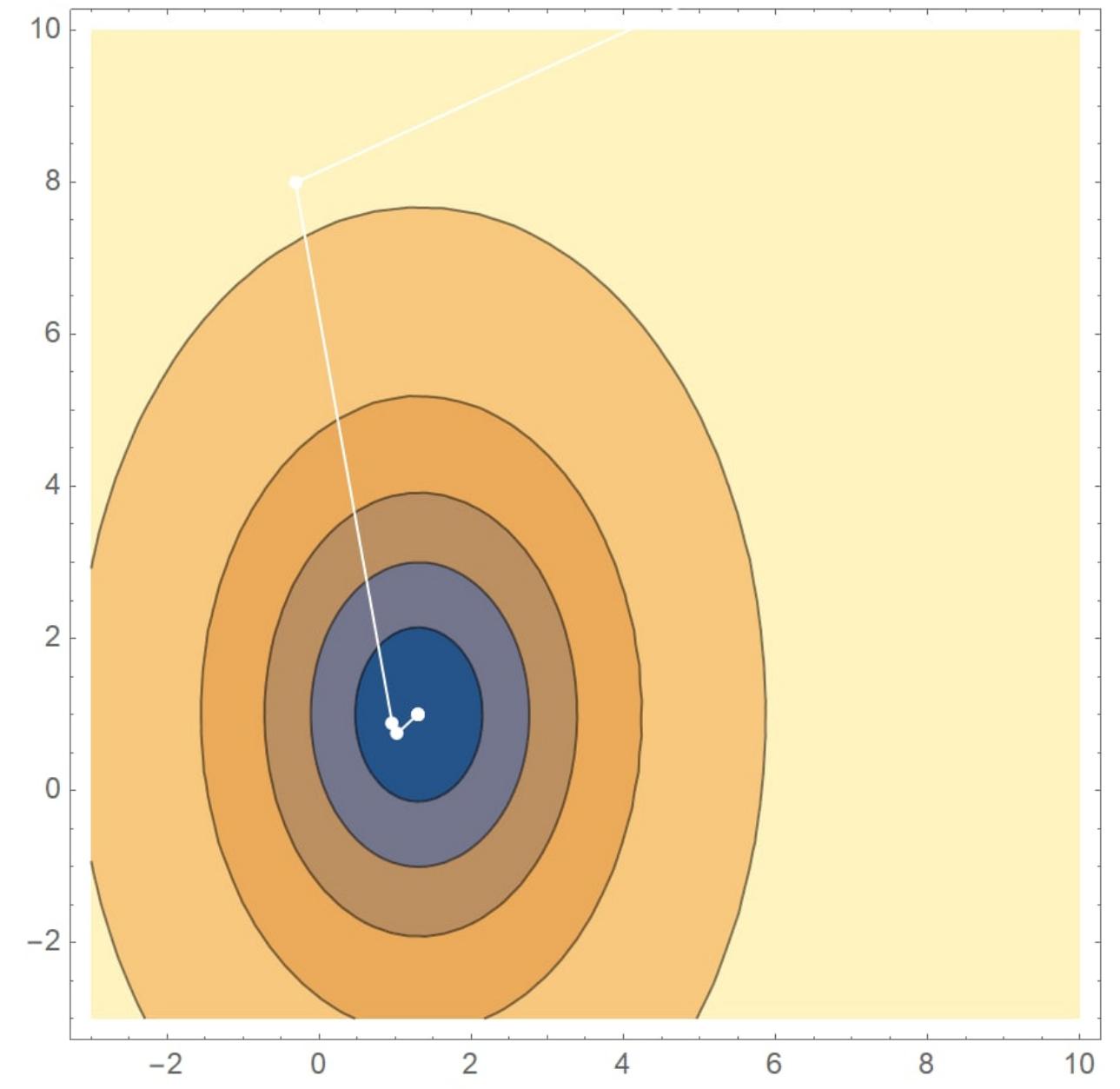
Начальное приближение:  $(x, y) = (0.0, 0.0)$



Начальное приближение:  $(x, y) = (10.0, 0.0)$



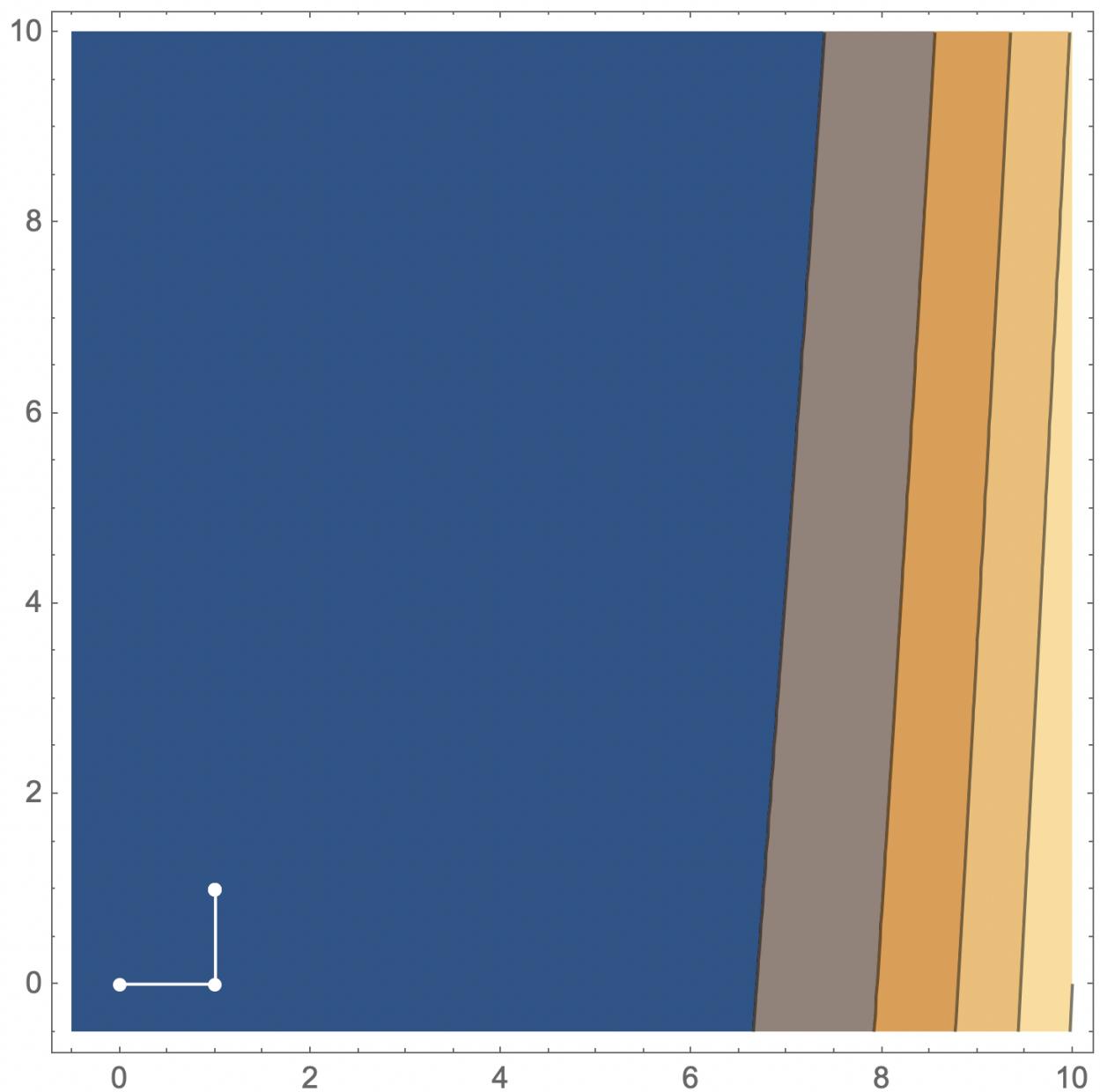
Начальное приближение:  $(x, y) = (15.0, 15.0)$



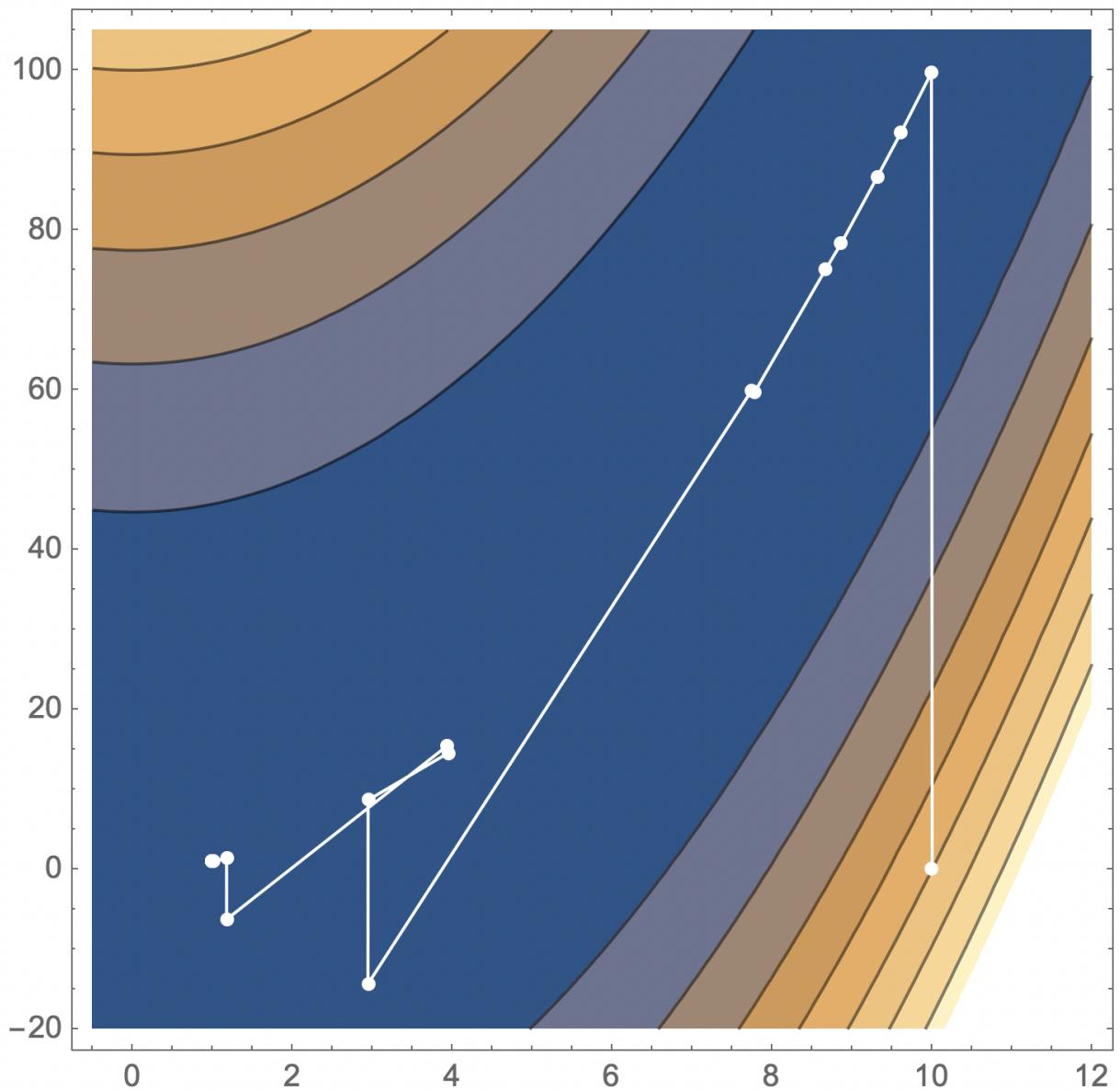
### 5.3. Классический Ньютона

$$5.3.1. \quad f(x) = 100 \cdot (x_2 - x_1^2)^2 + (1 - x_1)^2$$

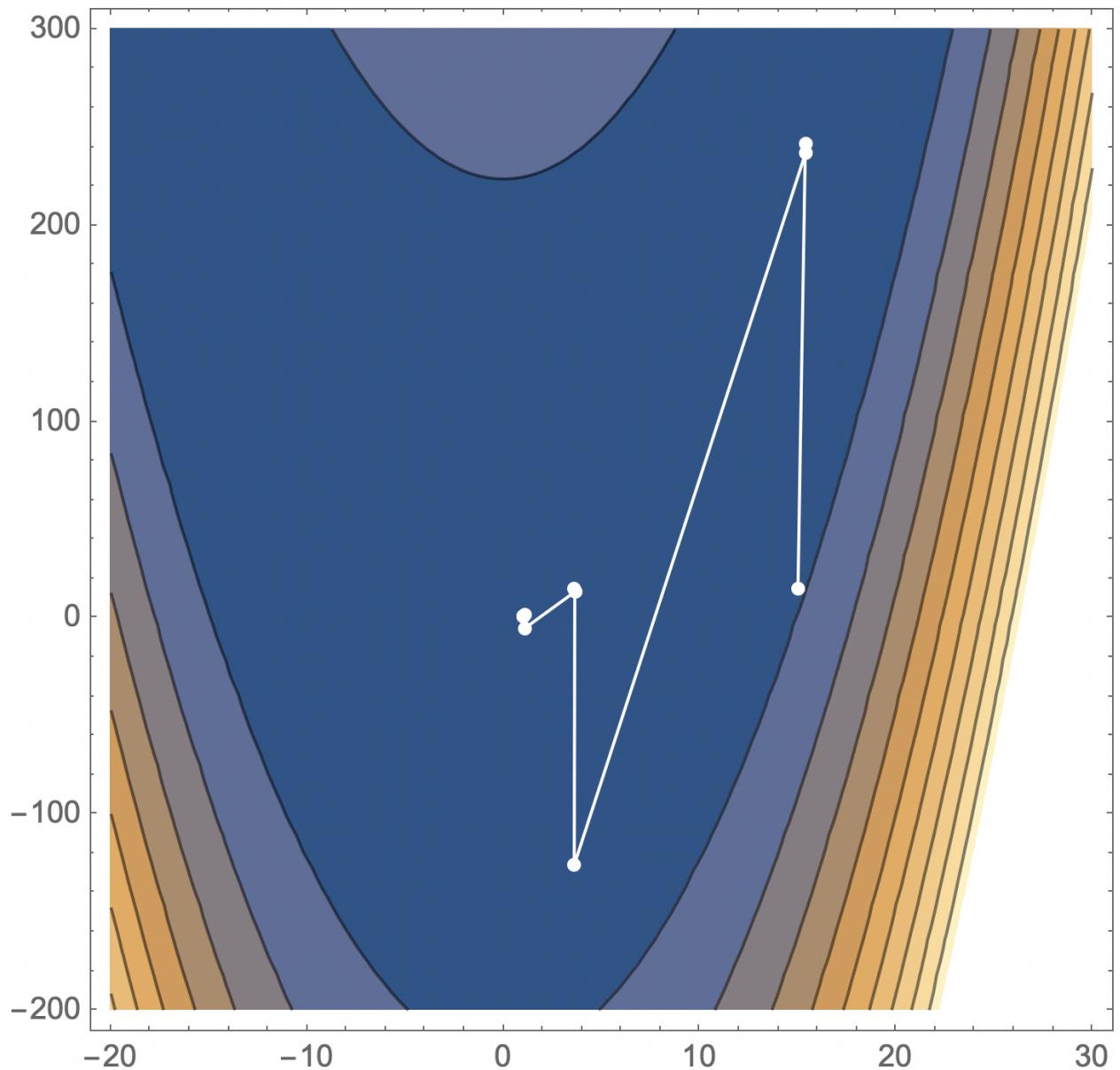
Начальное приближение:  $(x, y) = (0.0, 0.0)$



Начальное приближение:  $(x, y) = (10.0, 0.0)$

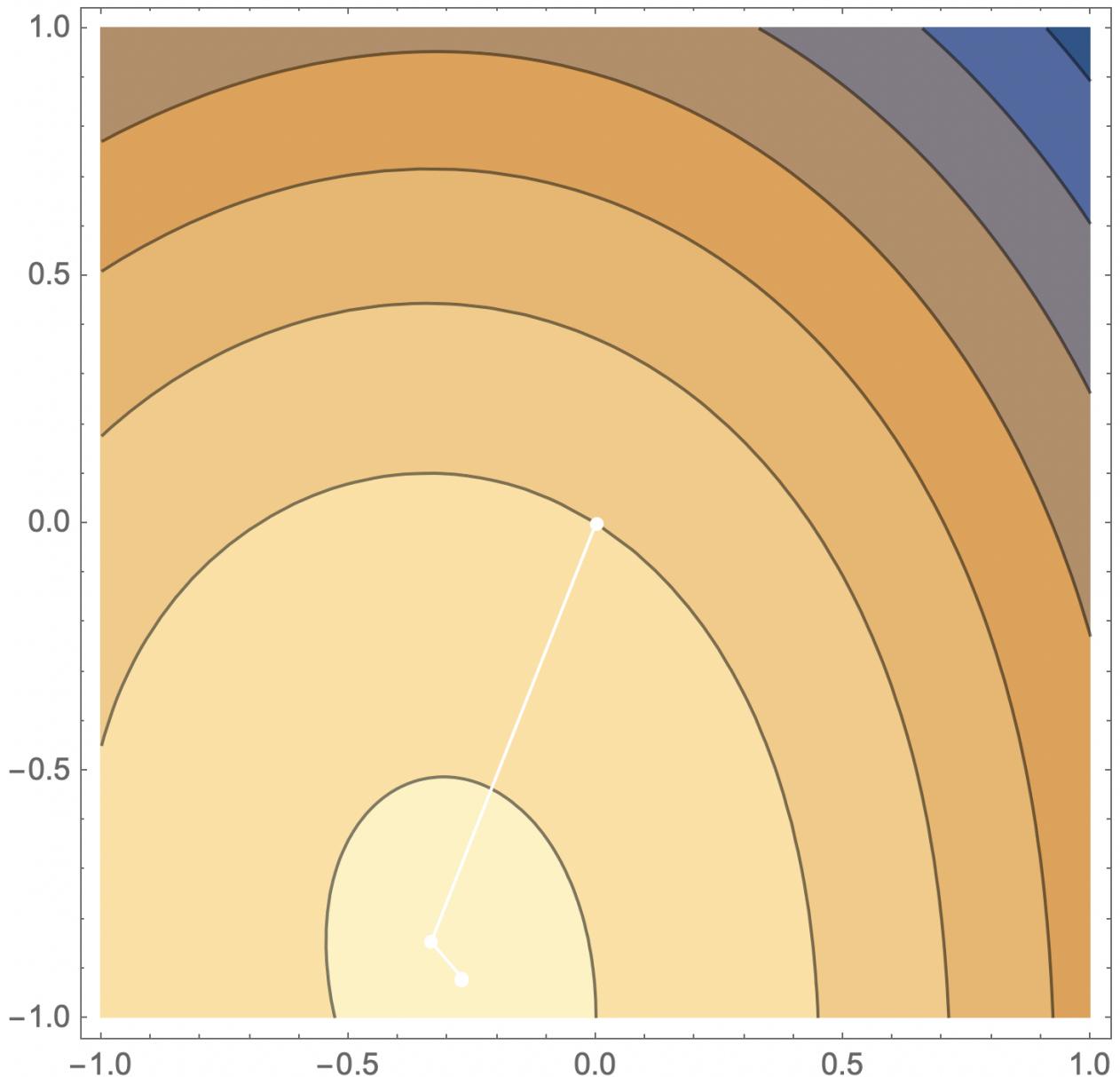


Начальное приближение:  $(x, y) = (15.0, 15.0)$

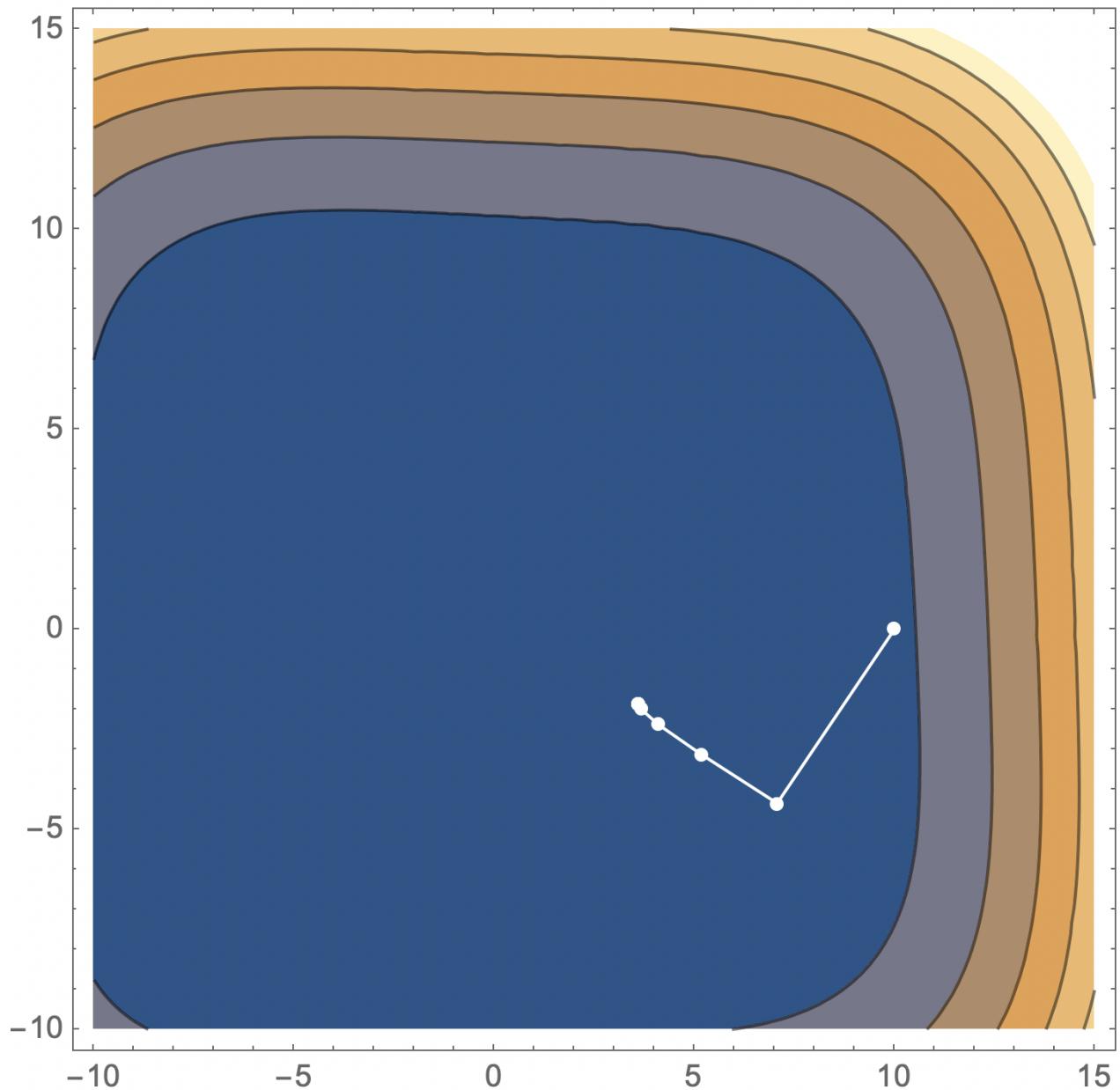


$$5.3.2. \ f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

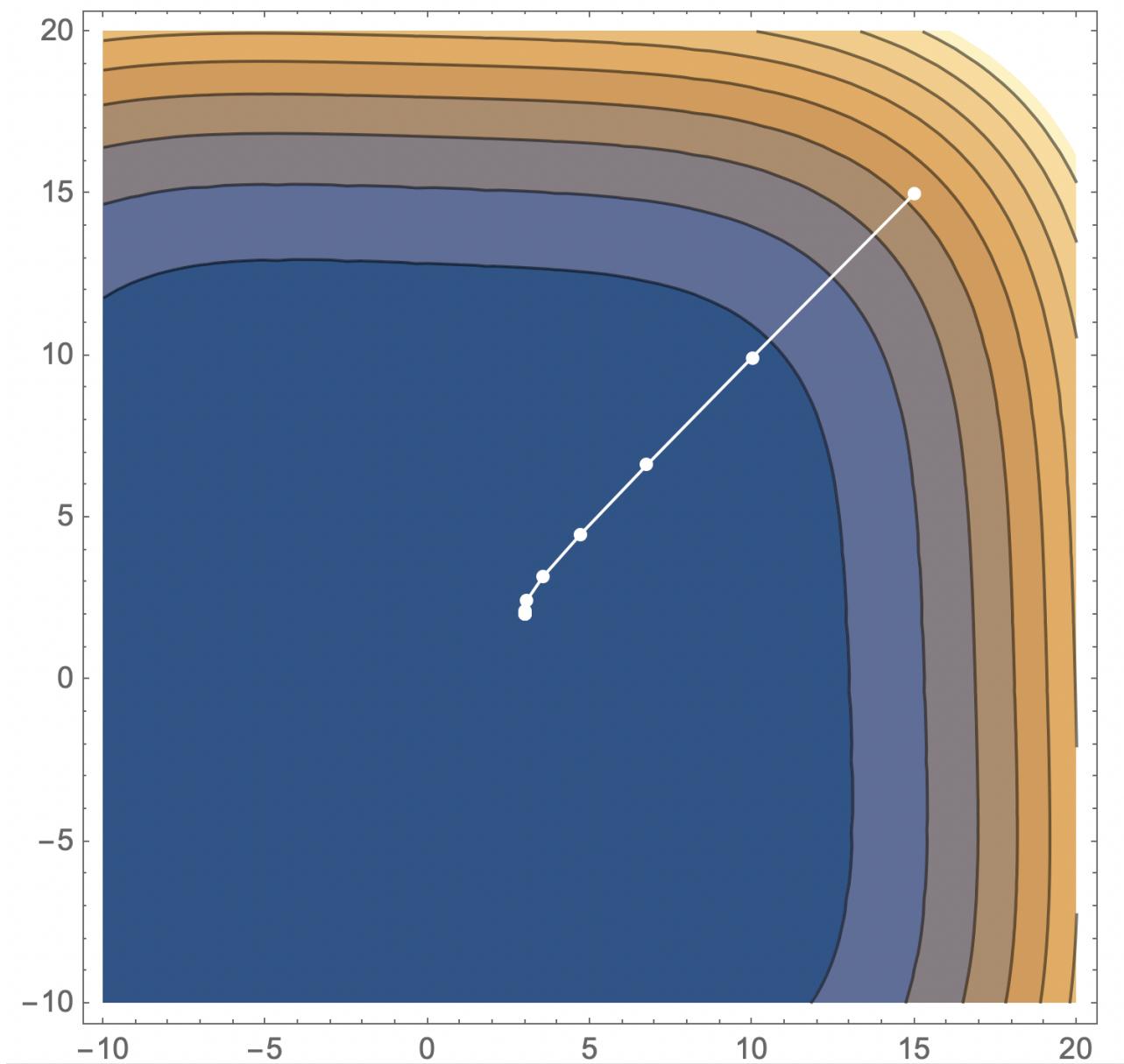
Начальное приближение:  $(x, y) = (0.0, 0.0)$ . В данном случае метод нашел максимум



Начальное приближение:  $(x, y) = (10.0, 0.0)$

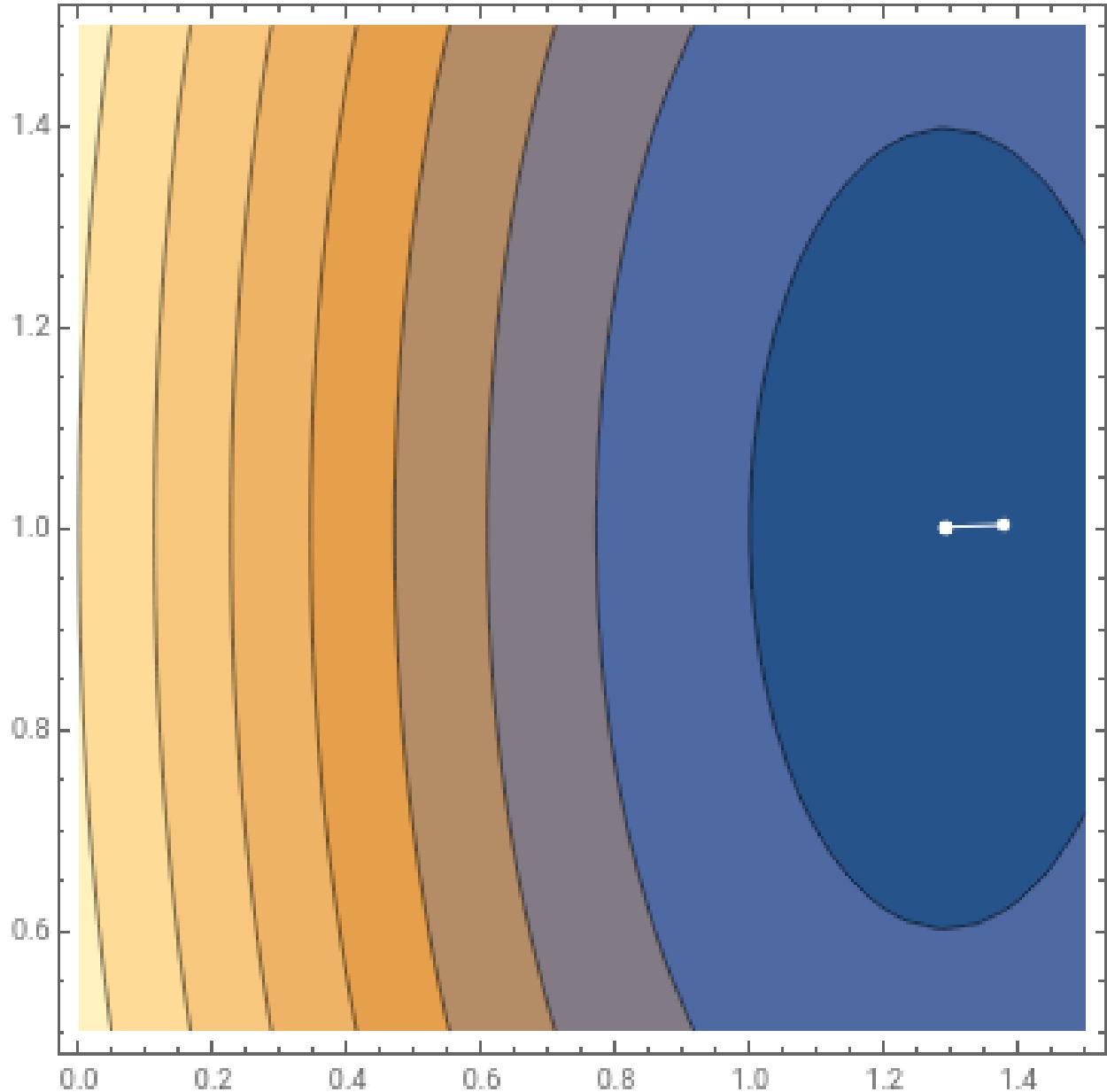


Начальное приближение:  $(x, y) = (15.0, 15.0)$

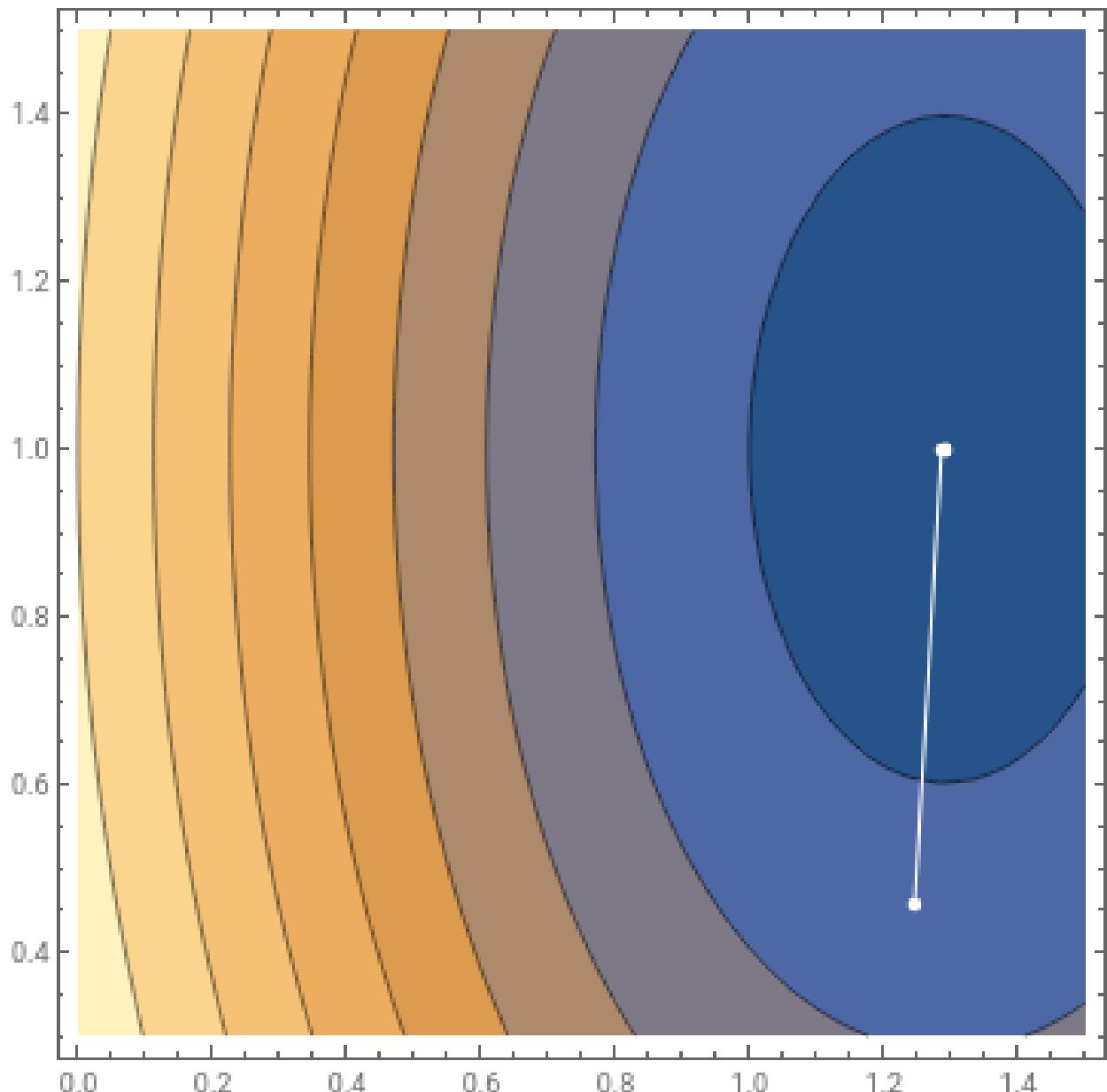


$$5.3.3. \quad f(x) = 100 - \frac{2}{1+(\frac{x_1-1}{2})^2 + (\frac{x_2-1}{3})^2} - \frac{1}{1+(\frac{x_1-2}{2})^2 + (\frac{x_2-1}{3})^2}$$

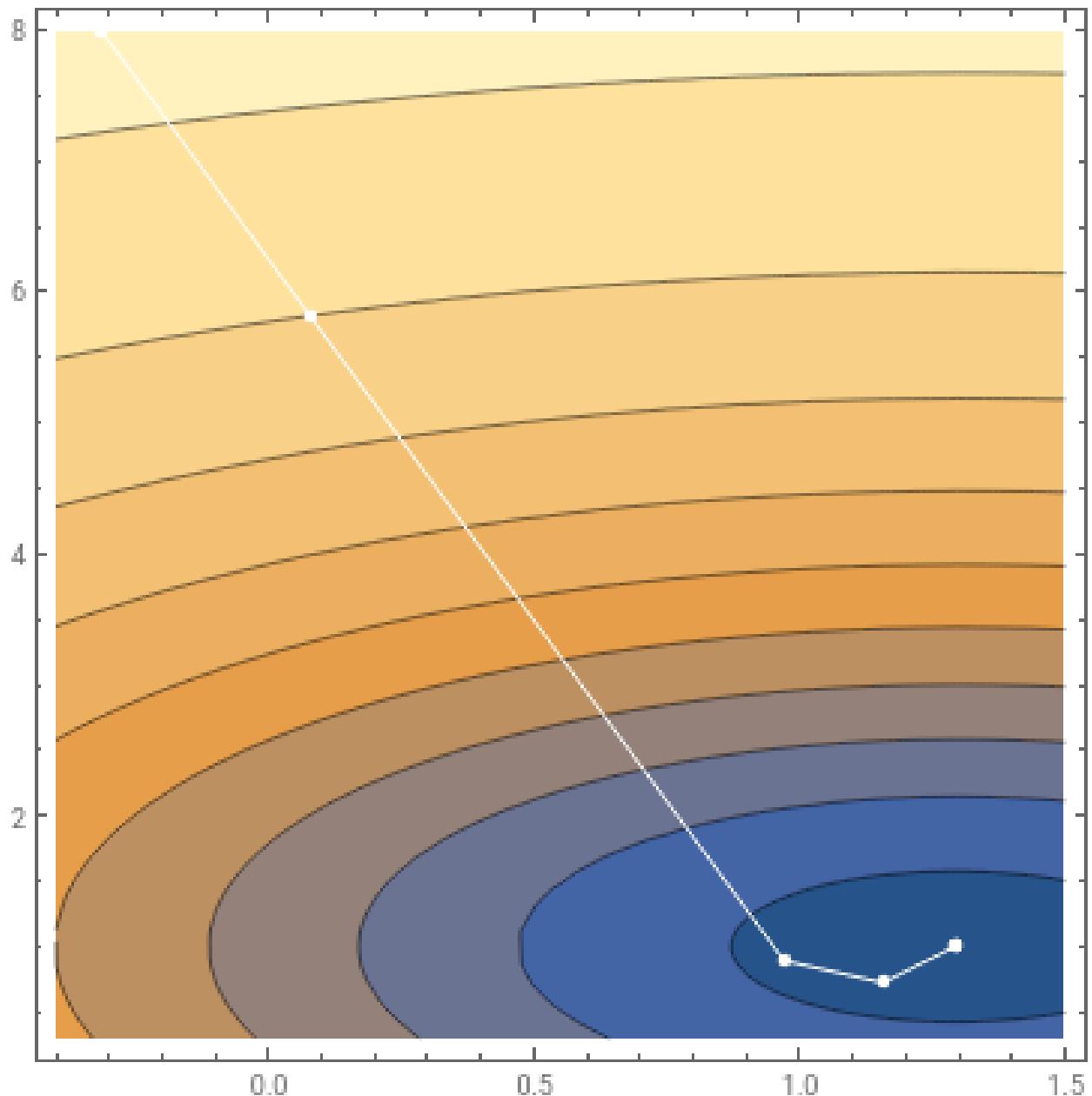
Начальное приближение:  $(x, y) = (0.0, 0.0)$



Начальное приближение:  $(x, y) = (10.0, 0.0)$



Начальное приближение:  $(x, y) = (15.0, 15.0)$



$$5.3.4. \quad f(x) = (x_1 + 10 \cdot x_2)^2 + 5 \cdot (x_3 - x_4)^2 + (x_2 - 2 \cdot x_3)^4 + 10 \cdot (x_1 - x_4)^6$$

Ниже приведены таблицы итераций для каждого из исследуемых выше методов. Возможности построения траектории для функции в четырехмерном пространстве в ходе исследования не было.

#### ДФП

Начальное приближение	Количество итераций
(0, 0, 0, 0)	0
(0, 10, 0, 10)	83
(15, 15, 15, 15)	21

#### Пауэлл

Начальное приближение	Количество итераций
(0, 0, 0, 0)	0
(0, 10, 0, 10)	25
(15, 15, 15, 15)	15

#### Классический Ньютон

Начальное приближение	Количество итераций
(0, 0, 0, 0)	13
(0, 10, 0, 10)	22
(15, 15, 15, 15)	29

### 5.4. Сравнение методов

В таблице приведены результаты запусков методов на начальных точках (0, 0), (10, 0) и (15, 15). Знак вопроса означает, что метод не сошелся.

Функция	ДФП	Пауэлл	Ньютон
$f_1$	14 / 3005 / 2596	15 / 76 / 91	5 / 19 / 11
$f_2$	8 / 6 / 5	8 / 6 / 5	5 / 8 / 10
$f_3$	0 / 83 / 21	0 / 25 / 15	? / 171194 / 67550
$f_4$	4 / 6 / 8	4 / 6 / 7	13 / 22 / 29

Методы ДФП и Пауэлла имеют одинаковую скорость сходимости. Результаты сильно зависят от выбора начального приближения, даже если точки приближения находятся на небольшом расстоянии.

Метод Ньютона выигрывает по скорости сходимости у ДФП и Пауэлла, однако во второй функции метод Ньютона сошелся к точке максимума, а не точке минимума.

Метод Ньютона показал плохую сходимость на функции  $f_4$ .

## 6. Программная реализация

### 6.1. Интерфейс методов Ньютона

Интерфейс объединяет в себе все методы Ньютона и, также, квазиньютоновские методы.

```
/*
 * Interface for newton methods.
 */
public interface NewtonMethod {

    /**
     * Find minimum.
     *
     * @return minimum.
     */
    Vector minimize();

    /**
     * Calculate alpha for current method. Default is 1.0.
     *
     * @param xk point.
     * @param pk direction.
     * @return alpha.
     */
    default double getAlpha(final Vector xk, final Vector pk) {
        return 1.0;
    }

    /**
     * Calculate direction for current method. Default is gauss
     * calculation.
     *
     * @param hesseMatrix Hessian.
     * @param gradient calculated gradient.
     * @param eps given epsilon.
     * @param defaultP default vector if no solution found.
     * @return calculated direction.
     */
    default Vector getDirection(final FullMatrix hesseMatrix, final
        Vector gradient, final double eps, final Vector defaultP) {
        return hesseMatrix.gauss(gradient.mul(-1), eps).
            orElse(defaultP);
    }
}
```

## 6.2. Абстрактный метод Ньютона

Предоставляет обобщенную функцию минимизации.

```
/**  
 * Abstract Newton method.  
 */  
public abstract class AbstractNewtonMethod implements NewtonMethod {  
  
    /**  
     * Given function.  
     */  
    protected final Function<Vector, Double> function;  
  
    /**  
     * Given epsilon.  
     */  
    protected final Double eps;  
  
    /**  
     * Start point.  
     */  
    protected final Vector startPoint;  
  
    /**  
     * Start point (vector) size.  
     */  
    protected final int size;  
  
    /**  
     * Iteration steps.  
     */  
    private final Steps steps;  
  
    public AbstractNewtonMethod(final Function<Vector, Double>  
        function, final Double eps, final Vector startPoint) {  
        this.function = function;  
        this.eps = eps;  
        this.startPoint = startPoint;  
        this.size = startPoint.size();  
        this.steps = new Steps(new ArrayList<>(), startPoint);  
    }  
  
    /**  
     * Gradient calculation for given point.  
     *  
     * @param vector given point.  
     * @return gradient.  
     */  
    protected Vector gradient(final Vector vector) {
```

```

        final double[] result = new double[size];
        final double f0 = function.apply(vector);
        for (int i = 0; i < size; i++) {
            final var curVector = vector.add(i, eps);
            result[i] = (function.apply(curVector) - f0) / eps;
        }
        return Vector.of(result);
    }

    /**
     * Hessian calculation for given point
     *
     * @param vector given point.
     * @return Hessian.
     */
    protected double[][][] hesseMatrixCalculation(final Vector vector)
    {
        final double[][][] result = new double[size][size];
        final double f0 = function.apply(vector);
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                final double fij = function.apply(vector.add(i, eps)
.add(j, eps));
                final double fi = function.apply(vector.add(i, eps));
;
                final double fj = function.apply(vector.add(j, eps));
;
                result[i][j] = (fij - fi - fj + f0) / (eps * eps);
            }
        }
        return result;
    }

    /**
     * Minimization.
     *
     * @return minimum.
     */
    @Override
    public Vector minimize() {
        Vector xPrev = startPoint;
        while (true) {
            final var gradient = gradient(xPrev);
            final var H = (new FullMatrix(hesseMatrixCalculation(
xPrev)));
            final var pk = getDirection(H, gradient, eps, gradient.
mul(-1.0));
            final double alpha = getAlpha(xPrev, pk);
            final var xK = xPrev.add(pk.mul(alpha));

```

```
        steps.addIteration(alpha, xK, pk, function.apply(xK));
        if (steps.size() > 100000) throw new
IllegalArgumentException("Too much iterations");
        if (xK.sub(xPrev).norm() < eps) {
            xPrev = xK;
            break;
        }
        xPrev = xK;
    }
    return xPrev;
}

/**
 * Iteration steps.
 *
 * @return iteration steps.
 */
public Steps getSteps() {
    return steps;
}
}
```

### 6.3. Абстрактный квазиньютоновский метод

Предоставляет обобщенную функцию минимизации.

```
/*
 * Abstract quasi-Newton method.
 */
public abstract class AbstractQuasiNewtonMethod extends
OneDirectionNewtonMethod {
    public AbstractQuasiNewtonMethod( final Function<Vector, Double>
function, final Double eps, final Vector startPoint) {
        super(function, eps, startPoint);
    }

    /**
     * Minimize.
     *
     * @return point of minimum.
     */
    @Override
    public Vector minimize() {
        int iteration = 0;
        var E = FullMatrix.identityMatrix(size);
        FullMatrix prevG = E;
        Vector prevW = gradient(startPoint).mul(-1.0);
        Vector prevP = prevW;
        double ak = getAlpha(startPoint, prevP);
        Vector prevX = startPoint.add(prevP.mul(ak));
        Vector prevDX = prevX.sub(startPoint);
        while (prevDX.norm() > eps) {
            Vector nextW = gradient(prevX).mul(-1.0);
            FullMatrix nextG = iteration > size ? E : generateG(
prevX, prevW, prevG, prevDX, nextW);
            Vector nextP = nextG.multiply(nextW);
            ak = getAlpha(prevX, nextP);
            Vector nextX = prevX.add(nextP.mul(ak));
            steps.addIteration(ak, nextX, nextP, function.apply(
nextX));

            prevG = nextG;
            prevW = nextW;
            prevDX = nextX.sub(prevX);
            prevX = nextX;
            iteration++;
        }
        return prevX;
    }

    /**
     * Method special G matrix generation
     *
```

```
* @param prevX previous point.
* @param prevW previous W vector.
* @param prevG previous G matrix.
* @param prevDX previous DX vector.
* @param nextW nextW vector.
* @return new G_k.
*/
protected abstract FullMatrix generateG(
    final Vector prevX,
    final Vector prevW,
    final FullMatrix prevG,
    final Vector prevDX,
    final Vector nextW
);
}
```

## 6.4. Классический метод Ньютона

```
/**  
 * Classic Newton method.  
 */  
public class ClassicNewtonMethod extends AbstractNewtonMethod {  
  
    public ClassicNewtonMethod(final Function<Vector, Double>  
        function, final Double eps, final Vector startPoint) {  
        super(function, eps, startPoint);  
    }  
}
```

## 6.5. Метод ньютона с одномерной оптимизацией методом золотого сечения

```
/**  
 * One direction Newton method with golden ratio for alpha_k  
 * generation.  
 */  
public class OneDirectionNewtonMethod extends AbstractNewtonMethod {  
    public OneDirectionNewtonMethod(final Function<Vector, Double>  
        function, final Double eps, final Vector startPoint) {  
        super(function, eps, startPoint);  
    }  
  
    /**  
     * Generates alpha_k with golden ration method.  
     * @param xk point.  
     * @param pk direction.  
     * @return generated alpha_k.  
     */  
    @Override  
    public double getAlpha(final Vector xk, final Vector pk) {  
        var method = new GoldenRatioMethod(-1000d, 1000d, x ->  
            function.apply(xk.add(pk.mul(x))), eps);  
        method.calculate();  
        return method.getMinArgument();  
    }  
}
```

## 6.6. Метод ньютона с направлением спуска

```
/**  
 * Descend Newton method.  
 */  
public class DescendMethod extends OneDirectionNewtonMethod {  
    public DescendMethod(final Function<Vector, Double> function,  
                        final Double eps, final Vector startPoint) {  
        super(function, eps, startPoint);  
    }  
  
    /**  
     * Generates direction relative to the gradient sign.  
     *  
     * @param hesseMatrix Hessian.  
     * @param gradient calculated gradient.  
     * @param eps given epsilon.  
     * @param defaultP default vector if no solution found.  
     * @return generated direction.  
     */  
    @Override  
    public Vector getDirection(final FullMatrix hesseMatrix, final  
        Vector gradient, final double eps, final Vector defaultP) {  
        final var minusGradient = gradient.mul(-1);  
        final var p = super.getDirection(hesseMatrix, gradient, eps,  
            defaultP);  
        final var scalar = p.scalarMultiply(gradient);  
        return scalar > 0 ? minusGradient : p;  
    }  
}
```

## 6.7. Метод Давидона-Флетчера-Пауэлла

```
/**  
 * Davidon–Fletcher–Powell method.  
 */  
public class DavidonFletcherPowellMethod extends  
AbstractQuasiNewtonMethod {  
    public DavidonFletcherPowellMethod(final Function<Vector, Double  
> function, final Double eps, final Vector startPoint) {  
        super(function, eps, startPoint);  
    }  
  
    /**  
     * Davidon–Fletcher–Powell method special G matrix generation  
     *  
     * @param prevX previous point.  
     * @param prevW previous W vector.  
     * @param prevG previous G matrix.  
     * @param prevDX previous DX vector.  
     * @param nextW nextW vector.  
     * @return new G_k.  
     */  
    @Override  
    protected FullMatrix generateG(final Vector prevX, final Vector  
prevW, final FullMatrix prevG, final Vector prevDX, final Vector  
nextW) {  
        Vector dw = nextW.sub(prevW);  
        Vector v = prevG.multiply(dw);  
        return prevG.sub(prevDX.mulByTransposed(prevDX).div(dw.  
scalarMultiply(prevDX))).sub(v.mulByTransposed(v).div(v.  
scalarMultiply(dw)));  
    }  
}
```

## 6.8. Метод Пауэлла

```
/**  
 * Powell method.  
 */  
public class PowellMethod extends AbstractQuasiNewtonMethod {  
    public PowellMethod(final Function<Vector, Double> function,  
        final Double eps, final Vector startPoint) {  
        super(function, eps, startPoint);  
    }  
  
    /**  
     * Powell method special G matrix generation  
     *  
     * @param prevX previous point.  
     * @param prevW previous W vector.  
     * @param prevG previous G matrix.  
     * @param prevDX previous DX vector.  
     * @param nextW nextW vector.  
     * @return new G_k.  
     */  
    protected FullMatrix generateG(final Vector prevX, final Vector  
        prevW, final FullMatrix prevG, final Vector prevDX, final Vector  
        nextW) {  
        Vector dw = nextW.sub(prevW);  
        Vector dWaveX = prevDX.add(prevG.multiply(dw));  
        return prevG.sub(dWaveX.mulByTransposed(dWaveX).div(dw.  
            scalarMultiply(dWaveX)));  
    }  
}
```

## 7. Диаграмма классов

## 8. Выводы

В ходе лабораторной работы были реализованы методы Ньютона и его различные модификации, а также квазиньютоновские методы. Исследование было проведено как на квадратичных функциях, так и на не квадратичных. Было исследовано влияние выбора начального приближения на точность и скорость сходимости функции.

Методы Ньютона различаются по выбору направления и построению релаксационной последовательности, а также показывают различные и неоднозначные результаты на исследуемых функциях.

- Классический метод Ньютона показывает себя лучше остальных методов, если начальное приближение находится в маленькой окрестности искомой точки минимума. Однако, если функция не является квадратичной, то результат непредсказуем.
- Метод Ньютона с направлением спуска позволяет находить минимум и для не квадратичных функций, однако, может показывать медленную скорость сходимости.
- Квазиньютоновские методы показали значительно лучшие результаты при наличии рестартов. Это обусловлено тем, что при наличии рестартов они обладают глобальность сходимостью. Однако, Гессиан некоторых функций удовлетворял условию Липшица и на таких функциях скорость сходимости снижалась до квадратичной.