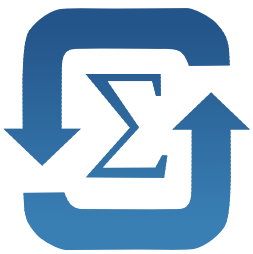
Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Лабораторная работа №2
по дисциплине «Методы оптимизации»

Методы спуска



Факультет:	ПМИ
Группа:	ПМ-63
Студенты:	Шепрут И.И. Крашенинник Н.А. Пешкичева А.А.
Вариант:	4
Преподаватель:	Чимитова Е.В.

Новосибирск
2019

1 Цель работы

Ознакомиться с методами поиска минимума фукнции n переменных в оптимизационных задачах без ограничений.

2 Задание

• Реализовать два метода поиска экстремума функции: **метод Бройдена**, **метод Сопряженных Градиентов в модификации Флетчера-Ривса**.
• Исследовать алгоритмы на квадратичной функции, на функции Розенброка и на заданной по варианту функции. Исследовать в зависимости от различной точности и начального приближения.
• Построить траекторию спуска различных алгоритмов, наложить эту траекторию на рисунок с линиями равного уровня заданной функции.
• Реализовать метод парабол для одномерного поиска. Исследовать влияние точности одномерного поиска на общее количество итераций и вычислений функции при разных методах одномерного поиска.

3 Таблицы и визуализация

Далее для всех пунктов: $\mathbf{x}_0 = (0,0)^T$; если явно не указано, то $\varepsilon = 0.001$; если явно не указано, то метод одномерного поиска: поиск отрезка из предыдущей лабораторной работы + метод золотого сечения.

Три изображения в ряд показывают следующее: на первом изображении — траекторию сходимости; на втором и третьем изображении — зависимость числа вычислений функции от начального приближения для различных методов. На первом изображении показны изолинии функции вместе с траекторией сходимости каждого метода. На втором и третьем изображении показана та же область, что и на первой, только градиентом показано число вычислений функции для сходимости метода. Цвет ближе к белому означает, что требуется меньше вычислений функции, ближе к черному, что больше. Для второй и третьей картинки показано максимальное, минимальное и среднее число вычислений функции для сходимости на этой области.

На основании этих изображений можно делать выводы о характере сходимости метода в зависимости от начального приближения.

3.1 Квадратичная функция

Функция $100(y-x)^2 + (1-x)^2$.

3.1.1 Первая таблица

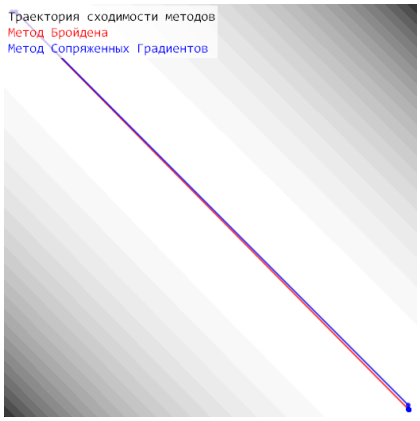
Метод Бройдена

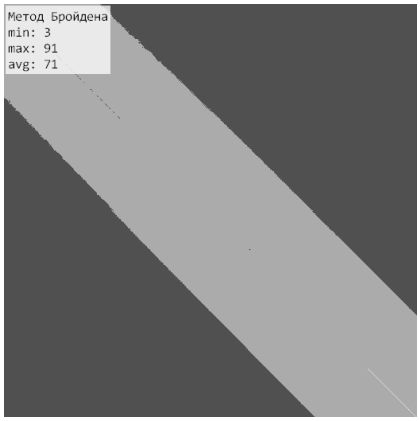
$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	2	55	$3.41 \cdot 10^{-4}$	$1.53 \cdot 10^{-8}$
-4	2	64	$1.43 \cdot 10^{-5}$	$3.59 \cdot 10^{-12}$
-5	2	74	$1.42 \cdot 10^{-6}$	$6.39 \cdot 10^{-14}$
-6	3	123	$8.14 \cdot 10^{-6}$	$9.1 \cdot 10^{-14}$

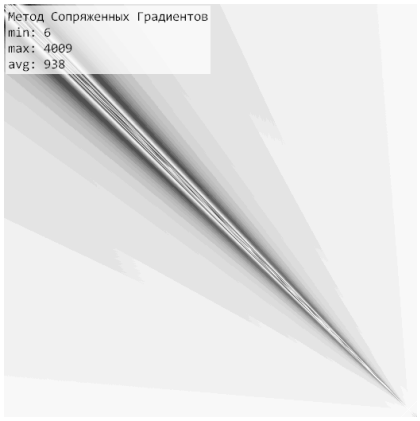
Метод Сопряженных Градиентов

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	22	973	$1.02 \cdot 10^{-2}$	$2.35 \cdot 10^{-7}$
-4	46	2,029	$9.75 \cdot 10^{-4}$	$2.15 \cdot 10^{-9}$
-5	70	3,085	$9.47 \cdot 10^{-5}$	$2.1 \cdot 10^{-11}$
-6	100	4,405	$6.38 \cdot 10^{-6}$	$1.19 \cdot 10^{-13}$

3.1.2 Изображения







3.1.3 Вторая таблица для метода Бройдена

i	(x, y)	$f(x, y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x, y)$	$H(f)$
0	$(0, 0)^T$	1	$(0, 0)^T$	0	$(0, 0)^T$	0	$(-2, 0)^T$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
1	$(0.97 \cdot 10^{-2}, 0)^T$	0.99	$(-2, 0)^T$	$4.85 \cdot 10^{-3}$	$(0.97 \cdot 10^{-2}, 0)^T$	$9.9 \cdot 10^{-3}$	$(-0.04, -1.9)^T$	$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$
2	$(1, 1)^T$	$1.53 \cdot 10^{-8}$	$(-0.99, -1)^T$	1	$(0.99, 1)^T$	0.99	$(0.85 \cdot 10^{-4}, -0.33 \cdot 10^{-3})^T$	$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.51 \end{pmatrix}$

3.1.4 Вторая таблица для метода сопряженных градиентов

i	(x, y)	$f(x, y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x, y)$	$H(f)$
0	$(0, 0)^T$	1	$(0, 0)^T$	0	$(0, 0)^T$	0	$(-2, 0)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
1	$(0.99 \cdot 10^{-2}, 0)^T$	0.99	$(2, -0)^T$	$4.95 \cdot 10^{-3}$	$(0.99 \cdot 10^{-2}, 0)^T$	$9.9 \cdot 10^{-3}$	$(-2, 0)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
2	$(1, 0.99)^T$	$9.8 \cdot 10^{-3}$	$(2, 2)^T$	0.5	$(0.99, 0.99)^T$	0.98	$(0.56 \cdot 10^{-6}, -2)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
3	$(1, 1)^T$	$4.2 \cdot 10^{-6}$	$(0.82, 4.8)^T$	$2.5 \cdot 10^{-3}$	$(0.2 \cdot 10^{-2}, 0.012)^T$	$9.8 \cdot 10^{-3}$	$(2, -2)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
4	$(1, 1)^T$	$4.16 \cdot 10^{-6}$	$(-0.5 \cdot 10^{-2}, 0.85 \cdot 10^{-3})^T$	$3.72 \cdot 10^{-3}$	$(-0.18 \cdot 10^{-4}, 0.32 \cdot 10^{-5})^T$	$4.7 \cdot 10^{-8}$	$(0.5 \cdot 10^{-2}, -0.85 \cdot 10^{-3})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
5	$(1, 1)^T$	$3.92 \cdot 10^{-6}$	$(-0.41 \cdot 10^{-2}, -0.29 \cdot 10^{-2})^T$	$3.89 \cdot 10^{-2}$	$(-0.16 \cdot 10^{-3}, -0.11 \cdot 10^{-3})^T$	$2.41 \cdot 10^{-7}$	$(0.6 \cdot 10^{-3}, 0.35 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
6	$(1, 1)^T$	$3.37 \cdot 10^{-6}$	$(-0.9 \cdot 10^{-2}, -0.025)^T$	$4.48 \cdot 10^{-3}$	$(-0.4 \cdot 10^{-4}, -0.11 \cdot 10^{-3})^T$	$5.47 \cdot 10^{-7}$	$(-0.9 \cdot 10^{-2}, 0.013)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
7	$(1, 1)^T$	$3.31 \cdot 10^{-6}$	$(-0.57 \cdot 10^{-2}, 0.2 \cdot 10^{-2})^T$	$3.05 \cdot 10^{-3}$	$(-0.17 \cdot 10^{-4}, 0.61 \cdot 10^{-5})^T$	$5.53 \cdot 10^{-8}$	$(0.57 \cdot 10^{-2}, -0.2 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
8	$(1, 1)^T$	$3.27 \cdot 10^{-6}$	$(-0.36 \cdot 10^{-2}, -0.17 \cdot 10^{-2})^T$	$1.08 \cdot 10^{-2}$	$(-0.39 \cdot 10^{-4}, -0.19 \cdot 10^{-4})^T$	$4.38 \cdot 10^{-8}$	$(0.95 \cdot 10^{-3}, 0.27 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
9	$(1, 1)^T$	$2.96 \cdot 10^{-6}$	$(-0.64 \cdot 10^{-2}, -0.011)^T$	$1.11 \cdot 10^{-2}$	$(-0.71 \cdot 10^{-4}, -0.13 \cdot 10^{-3})^T$	$3.15 \cdot 10^{-7}$	$(-0.32 \cdot 10^{-2}, 0.68 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
10	$(1, 1)^T$	$2.85 \cdot 10^{-6}$	$(-0.77 \cdot 10^{-2}, 0.43 \cdot 10^{-2})^T$	$2.69 \cdot 10^{-3}$	$(-0.21 \cdot 10^{-4}, 0.12 \cdot 10^{-4})^T$	$1.06 \cdot 10^{-7}$	$(0.77 \cdot 10^{-2}, -0.43 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
11	$(1, 1)^T$	$2.83 \cdot 10^{-6}$	$(-0.34 \cdot 10^{-2}, -0.95 \cdot 10^{-3})^T$	$5.14 \cdot 10^{-3}$	$(-0.17 \cdot 10^{-4}, -0.49 \cdot 10^{-5})^T$	$1.57 \cdot 10^{-8}$	$(0.12 \cdot 10^{-2}, 0.22 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
12	$(1, 1)^T$	$2.09 \cdot 10^{-6}$	$(-0.53 \cdot 10^{-2}, -0.65 \cdot 10^{-2})^T$	$6.42 \cdot 10^{-2}$	$(-0.34 \cdot 10^{-3}, -0.42 \cdot 10^{-3})^T$	$7.48 \cdot 10^{-7}$	$(-0.13 \cdot 10^{-2}, 0.46 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
13	$(1, 1)^T$	$1.68 \cdot 10^{-6}$	$(-0.014, 0.011)^T$	$2.52 \cdot 10^{-3}$	$(-0.35 \cdot 10^{-4}, 0.28 \cdot 10^{-4})^T$	$4.05 \cdot 10^{-7}$	$(0.014, -0.011)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
14	$(1, 1)^T$	$1.68 \cdot 10^{-6}$	$(-0.26 \cdot 10^{-2}, -0.27 \cdot 10^{-3})^T$	$3.12 \cdot 10^{-3}$	$(-0.81 \cdot 10^{-5}, -0.85 \cdot 10^{-6})^T$	$5.29 \cdot 10^{-9}$	$(0.12 \cdot 10^{-2}, 0.14 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
15	$(1, 1)^T$	$1.18 \cdot 10^{-6}$	$(-0.38 \cdot 10^{-2}, -0.33 \cdot 10^{-2})^T$	0.12	$(-0.45 \cdot 10^{-3}, -0.39 \cdot 10^{-3})^T$	$4.96 \cdot 10^{-7}$	$(-0.3 \cdot 10^{-3}, 0.29 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
16	$(1, 1)^T$	$7.59 \cdot 10^{-7}$	$(0.012, -0.014)^T$	$2.51 \cdot 10^{-3}$	$(0.3 \cdot 10^{-4}, -0.35 \cdot 10^{-4})^T$	$4.21 \cdot 10^{-7}$	$(-0.012, 0.014)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
17	$(1, 1)^T$	$7.57 \cdot 10^{-7}$	$(-0.11 \cdot 10^{-3}, -0.17 \cdot 10^{-2})^T$	$2.87 \cdot 10^{-3}$	$(-0.32 \cdot 10^{-6}, -0.5 \cdot 10^{-5})^T$	$2.18 \cdot 10^{-9}$	$(0.93 \cdot 10^{-3}, 0.81 \cdot 10^{-3})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
18	$(1, 1)^T$	$6.45 \cdot 10^{-7}$	$(-0.2 \cdot 10^{-2}, -0.25 \cdot 10^{-2})^T$	$6.41 \cdot 10^{-2}$	$(-0.13 \cdot 10^{-3}, -0.16 \cdot 10^{-3})^T$	$1.12 \cdot 10^{-7}$	$(0.19 \cdot 10^{-2}, -0.12 \cdot 10^{-3})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
19	$(1, 1)^T$	$5.2 \cdot 10^{-7}$	$(-0.77 \cdot 10^{-2}, 0.63 \cdot 10^{-2})^T$	$2.52 \cdot 10^{-3}$	$(-0.2 \cdot 10^{-4}, 0.16 \cdot 10^{-4})^T$	$1.25 \cdot 10^{-7}$	$(0.77 \cdot 10^{-2}, -0.63 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
20	$(1, 1)^T$	$5.19 \cdot 10^{-7}$	$(-0.14 \cdot 10^{-2}, -0.15 \cdot 10^{-3})^T$	$3.12 \cdot 10^{-3}$	$(-0.45 \cdot 10^{-5}, -0.47 \cdot 10^{-6})^T$	$1.64 \cdot 10^{-9}$	$(0.64 \cdot 10^{-3}, 0.8 \cdot 10^{-3})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
21	$(1, 1)^T$	$3.65 \cdot 10^{-7}$	$(-0.21 \cdot 10^{-2}, -0.18 \cdot 10^{-2})^T$	0.12	$(-0.25 \cdot 10^{-3}, -0.22 \cdot 10^{-3})^T$	$1.54 \cdot 10^{-7}$	$(-0.17 \cdot 10^{-3}, 0.16 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
22	$(1, 1)^T$	$2.35 \cdot 10^{-7}$	$(0.67 \cdot 10^{-2}, -0.77 \cdot 10^{-2})^T$	$2.51 \cdot 10^{-3}$	$(0.17 \cdot 10^{-4}, -0.19 \cdot 10^{-4})^T$	$1.3 \cdot 10^{-7}$	$(-0.67 \cdot 10^{-2}, 0.77 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

3.2 Функция Розенброка

Функция $100(y - x^2)^2 + (1 - x)^2$

3.2.1 Первая таблица

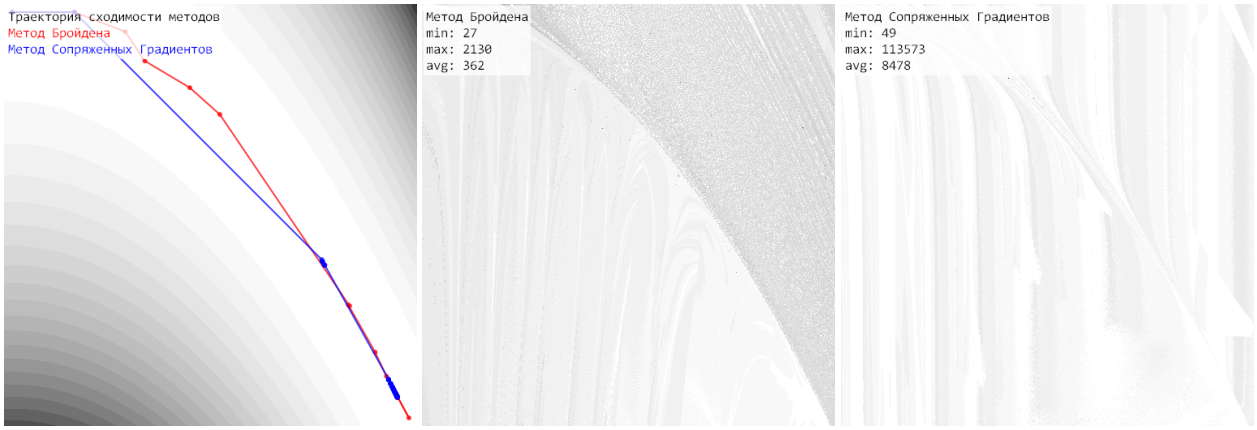
Метод Бройдена

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x, y)\|\|_{L_2}$	$f(x, y)$
-3	14	380	$1.87 \cdot 10^{-5}$	$3.41 \cdot 10^{-13}$
-4	13	409	$4.47 \cdot 10^{-5}$	$1.18 \cdot 10^{-12}$
-5	14	510	$1.94 \cdot 10^{-6}$	$8.93 \cdot 10^{-14}$
-6	14	579	$1.91 \cdot 10^{-6}$	$8.94 \cdot 10^{-14}$

Метод Сопряженных Градиентов

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x, y)\|\|_{L_2}$	$f(x, y)$
-3	112	4,933	$1.37 \cdot 10^{-2}$	$4.31 \cdot 10^{-7}$
-4	160	7,045	$1.97 \cdot 10^{-3}$	$2 \cdot 10^{-9}$
-5	166	7,309	$6.34 \cdot 10^{-4}$	$2.17 \cdot 10^{-11}$
-6	238	10,477	$1.15 \cdot 10^{-6}$	$5.14 \cdot 10^{-13}$

3.2.2 Изображения



3.2.3 Вторая таблица для метода Бройдена

i	(x, y)	$f(x, y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x, y)$	$H(f)$
0	$(0, 0)^T$	1	$(0, 0)^T$	0	$(0, 0)^T$	0	$(-2, 0)^T$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
1	$(0.16, 0)^T$	0.77	$(-2, 0)^T$	$8.09 \cdot 10^{-2}$	$(0.16, 0)^T$	0.23	$(0.015, -5.2)^T$	$\begin{pmatrix} 0.89 & 0.31 \\ 0.31 & 0.12 \end{pmatrix}$
2	$(0.29, 0.051)^T$	0.62	$(-1.6, -0.62)^T$	$8.14 \cdot 10^{-2}$	$(0.13, 0.051)^T$	0.15	$(2.7, -7.1)^T$	$\begin{pmatrix} 0.074 & 0.038 \\ 0.038 & 0.028 \end{pmatrix}$
3	$(0.34, 0.13)^T$	0.44	$(-0.063, -0.095)^T$	0.8	$(0.051, 0.076)^T$	0.19	$(-2.4, 1.6)^T$	$\begin{pmatrix} 0.038 & 0.028 \\ 0.028 & 0.026 \end{pmatrix}$
4	$(0.46, 0.2)^T$	0.32	$(-0.046, -0.027)^T$	2.55	$(0.12, 0.069)^T$	0.12	$(2, -3.3)^T$	$\begin{pmatrix} 0.21 & 0.16 \\ 0.16 & 0.13 \end{pmatrix}$
5	$(0.54, 0.27)^T$	0.27	$(-0.13, -0.11)^T$	0.61	$(0.078, 0.069)^T$	$4.32 \cdot 10^{-2}$	$(4.4, -5)^T$	$\begin{pmatrix} 0.084 & 0.078 \\ 0.078 & 0.073 \end{pmatrix}$
6	$(0.88, 0.76)^T$	$1.75 \cdot 10^{-2}$	$(-0.013, -0.02)^T$	25.33	$(0.34, 0.5)^T$	0.26	$(1.3, -0.9)^T$	$\begin{pmatrix} 0.17 & 0.21 \\ 0.21 & 0.28 \end{pmatrix}$
7	$(0.87, 0.76)^T$	$1.65 \cdot 10^{-2}$	$(0.035, 0.027)^T$	$9.33 \cdot 10^{-2}$	$(-0.33 \cdot 10^{-2}, -0.25 \cdot 10^{-2})^T$	$1.04 \cdot 10^{-3}$	$(0.2, -0.26)^T$	$\begin{pmatrix} 0.054 & 0.091 \\ 0.091 & 0.16 \end{pmatrix}$
8	$(0.94, 0.88)^T$	$5.73 \cdot 10^{-3}$	$(-0.013, -0.023)^T$	5.42	$(0.07, 0.12)^T$	$1.08 \cdot 10^{-2}$	$(1.7, -0.97)^T$	$\begin{pmatrix} 0.32 & 0.59 \\ 0.59 & 1.1 \end{pmatrix}$
9	$(0.97, 0.94)^T$	$1.08 \cdot 10^{-3}$	$(-0.02, -0.042)^T$	1.49	$(0.029, 0.062)^T$	$4.66 \cdot 10^{-3}$	$(-0.66, 0.31)^T$	$\begin{pmatrix} 1.5 \cdot 10^3 & 2.8 \cdot 10^3 \\ 2.8 \cdot 10^3 & 5.1 \cdot 10^3 \end{pmatrix}$
10	$(1, 1.1)^T$	$5.94 \cdot 10^{-3}$	$(-1.3 \cdot 10^2, -2.4 \cdot 10^2)^T$	$4.53 \cdot 10^{-4}$	$(0.058, 0.11)^T$	$4.86 \cdot 10^{-3}$	$(3, -1.4)^T$	$\begin{pmatrix} 0.23 & 0.46 \\ 0.46 & 0.9 \end{pmatrix}$
11	$(0.99, 0.98)^T$	$1.05 \cdot 10^{-4}$	$(0.046, 0.085)^T$	0.81	$(-0.038, -0.069)^T$	$5.84 \cdot 10^{-3}$	$(0.22, -0.12)^T$	$\begin{pmatrix} 0.19 & 0.37 \\ 0.37 & 0.74 \end{pmatrix}$
12	$(1, 1)^T$	$8.84 \cdot 10^{-6}$	$(-0.28 \cdot 10^{-2}, -0.6 \cdot 10^{-2})^T$	2.09	$(0.59 \cdot 10^{-2}, 0.013)^T$	$9.58 \cdot 10^{-5}$	$(-0.078, 0.037)^T$	$\begin{pmatrix} 0.37 & 0.75 \\ 0.75 & 1.5 \end{pmatrix}$
13	$(1, 1)^T$	$1.94 \cdot 10^{-8}$	$(-0.15 \cdot 10^{-2}, -0.29 \cdot 10^{-2})^T$	1.47	$(0.22 \cdot 10^{-2}, 0.43 \cdot 10^{-2})^T$	$8.82 \cdot 10^{-6}$	$(0.4 \cdot 10^{-2}, -0.21 \cdot 10^{-2})^T$	$\begin{pmatrix} 0.49 & 0.97 \\ 0.97 & 1.9 \end{pmatrix}$
14	$(1, 1)^T$	$3.41 \cdot 10^{-13}$	$(-0.83 \cdot 10^{-4}, -0.17 \cdot 10^{-3})^T$	1.12	$(0.93 \cdot 10^{-4}, 0.2 \cdot 10^{-3})^T$	$1.94 \cdot 10^{-8}$	$(0.17 \cdot 10^{-4}, -0.78 \cdot 10^{-5})^T$	$\begin{pmatrix} 0.51 & 1 \\ 1 & 2 \end{pmatrix}$

3.2.4 Вторая таблица для метода сопряженных градиентов

i	(x, y)	$f(x, y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x, y)$	$H(f)$
0	$(0, 0)^T$	1	$(0, 0)^T$	0	$(0, 0)^T$	0	$(-2, 0)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
1	$(0.16, 0)^T$	0.77	$(2, -0)^T$	$8.06 \cdot 10^{-2}$	$(0.16, 0)^T$	0.23	$(-2, 0)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
2	$(0.8, 0.64)^T$	$3.97 \cdot 10^{-2}$	$(5.2, 5.2)^T$	0.12	$(0.64, 0.64)^T$	0.73	$(-1 \cdot 10^{-6}, -5.2)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
3	$(0.8, 0.65)^T$	$3.84 \cdot 10^{-2}$	$(0.27, 1.6)^T$	$3.27 \cdot 10^{-3}$	$(0.88 \cdot 10^{-3}, 0.51 \cdot 10^{-2})^T$	$1.38 \cdot 10^{-3}$	$(0.65, -0.65)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
4	$(0.81, 0.65)^T$	$3.81 \cdot 10^{-2}$	$(0.54, -0.093)^T$	$1.62 \cdot 10^{-3}$	$(0.88 \cdot 10^{-3}, -0.15 \cdot 10^{-3})^T$	$2.44 \cdot 10^{-4}$	$(-0.54, 0.093)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
5	$(0.81, 0.65)^T$	$3.8 \cdot 10^{-2}$	$(0.26, 0.18)^T$	$4.51 \cdot 10^{-3}$	$(0.12 \cdot 10^{-2}, 0.82 \cdot 10^{-3})^T$	$1.11 \cdot 10^{-4}$	$(-0.038, -0.22)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
6	$(0.81, 0.66)^T$	$3.66 \cdot 10^{-2}$	$(0.3, 0.86)^T$	$1.01 \cdot 10^{-2}$	$(0.31 \cdot 10^{-2}, 0.87 \cdot 10^{-2})^T$	$1.39 \cdot 10^{-3}$	$(0.3, -0.43)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
7	$(0.81, 0.66)^T$	$3.6 \cdot 10^{-2}$	$(0.89, -0.32)^T$	$1.44 \cdot 10^{-3}$	$(0.13 \cdot 10^{-2}, -0.45 \cdot 10^{-3})^T$	$6.45 \cdot 10^{-4}$	$(-0.89, 0.32)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
8	$(0.81, 0.66)^T$	$3.59 \cdot 10^{-2}$	$(0.26, 0.12)^T$	$2.32 \cdot 10^{-3}$	$(0.6 \cdot 10^{-3}, 0.29 \cdot 10^{-3})^T$	$4.82 \cdot 10^{-5}$	$(-0.068, -0.19)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
9	$(0.98, 0.95)^T$	$7.47 \cdot 10^{-4}$	$(0.31, 0.55)^T$	0.54	$(0.17, 0.3)^T$	$3.52 \cdot 10^{-2}$	$(0.16, -0.33)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
10	$(0.98, 0.95)^T$	$5.71 \cdot 10^{-4}$	$(-0.51, 0.28)^T$	$1.04 \cdot 10^{-3}$	$(-0.53 \cdot 10^{-3}, 0.3 \cdot 10^{-3})^T$	$1.76 \cdot 10^{-4}$	$(0.51, -0.28)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
11	$(0.98, 0.95)^T$	$5.7 \cdot 10^{-4}$	$(-0.84 \cdot 10^{-2}, 0.03)^T$	$1.12 \cdot 10^{-3}$	$(-0.94 \cdot 10^{-5}, 0.33 \cdot 10^{-4})^T$	$2.66 \cdot 10^{-7}$	$(-0.011, -0.019)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
12	$(0.98, 0.95)^T$	$5.6 \cdot 10^{-4}$	$(0.018, 0.052)^T$	$1.91 \cdot 10^{-2}$	$(0.35 \cdot 10^{-3}, 1 \cdot 10^{-3})^T$	$9.76 \cdot 10^{-6}$	$(-0.031, -0.87 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
13	$(0.98, 0.95)^T$	$5.47 \cdot 10^{-4}$	$(0.15, -0.053)^T$	$1.06 \cdot 10^{-3}$	$(0.16 \cdot 10^{-3}, -0.56 \cdot 10^{-4})^T$	$1.36 \cdot 10^{-5}$	$(-0.15, 0.053)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
14	$(0.98, 0.95)^T$	$5.47 \cdot 10^{-4}$	$(0.027, 0.013)^T$	$1.41 \cdot 10^{-3}$	$(0.39 \cdot 10^{-4}, 0.18 \cdot 10^{-4})^T$	$3.24 \cdot 10^{-7}$	$(-0.72 \cdot 10^{-2}, -0.02)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
15	$(0.98, 0.96)^T$	$4.34 \cdot 10^{-4}$	$(0.03, 0.053)^T$	0.18	$(0.53 \cdot 10^{-2}, 0.95 \cdot 10^{-2})^T$	$1.13 \cdot 10^{-4}$	$(0.015, -0.032)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
16	$(0.98, 0.96)^T$	$3.4 \cdot 10^{-4}$	$(-0.37, 0.21)^T$	$1.03 \cdot 10^{-3}$	$(-0.39 \cdot 10^{-3}, 0.22 \cdot 10^{-3})^T$	$9.45 \cdot 10^{-5}$	$(0.37, -0.21)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
17	$(0.98, 0.96)^T$	$3.39 \cdot 10^{-4}$	$(-0.65 \cdot 10^{-2}, 0.023)^T$	$1.11 \cdot 10^{-3}$	$(-0.72 \cdot 10^{-5}, 0.25 \cdot 10^{-4})^T$	$1.56 \cdot 10^{-7}$	$(-0.81 \cdot 10^{-2}, -0.015)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
18	$(0.98, 0.96)^T$	$3.33 \cdot 10^{-4}$	$(0.014, 0.04)^T$	$1.95 \cdot 10^{-2}$	$(0.28 \cdot 10^{-3}, 0.78 \cdot 10^{-3})^T$	$5.9 \cdot 10^{-6}$	$(-0.024, -0.67 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
19	$(0.98, 0.96)^T$	$3.25 \cdot 10^{-4}$	$(0.12, -0.042)^T$	$1.05 \cdot 10^{-3}$	$(0.12 \cdot 10^{-3}, -0.43 \cdot 10^{-4})^T$	$8.17 \cdot 10^{-6}$	$(-0.12, 0.042)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
20	$(0.98, 0.96)^T$	$3.25 \cdot 10^{-4}$	$(0.021, 0.01)^T$	$1.39 \cdot 10^{-3}$	$(0.29 \cdot 10^{-4}, 0.14 \cdot 10^{-4})^T$	$1.89 \cdot 10^{-7}$	$(-0.55 \cdot 10^{-2}, -0.016)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
21	$(0.99, 0.97)^T$	$2.63 \cdot 10^{-4}$	$(0.023, 0.041)^T$	0.17	$(0.38 \cdot 10^{-2}, 0.68 \cdot 10^{-2})^T$	$6.17 \cdot 10^{-5}$	$(0.012, -0.024)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
22	$(0.99, 0.97)^T$	$2.1 \cdot 10^{-4}$	$(-0.28, 0.16)^T$	$1.02 \cdot 10^{-3}$	$(-0.29 \cdot 10^{-3}, 0.16 \cdot 10^{-3})^T$	$5.39 \cdot 10^{-5}$	$(0.28, -0.16)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
23	$(0.99, 0.97)^T$	$2.09 \cdot 10^{-4}$	$(-0.51 \cdot 10^{-2}, 0.018)^T$	$1.11 \cdot 10^{-3}$	$(-0.56 \cdot 10^{-5}, 0.2 \cdot 10^{-4})^T$	$9.53 \cdot 10^{-8}$	$(-0.64 \cdot 10^{-2}, -0.011)^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$
24	$(0.99, 0.97)^T$	$2.06 \cdot 10^{-4}$	$(0.011, 0.032)^T$	$1.98 \cdot 10^{-2}$	$(0.22 \cdot 10^{-3}, 0.63 \cdot 10^{-3})^T$	$3.69 \cdot 10^{-6}$	$(-0.019, -0.53 \cdot 10^{-2})^T$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$

Дальше таблица не показана в целях экономии места, так как она слишком большая.

3.3 Функция по варианту

Функция $\frac{2}{1+\left(\frac{x-1}{2}\right)^2+\left(\frac{y-2}{1}\right)^2} + \frac{1}{1+\left(\frac{x-3}{3}\right)^2+\left(\frac{y-1}{3}\right)^2}$.

3.3.1 Первая таблица

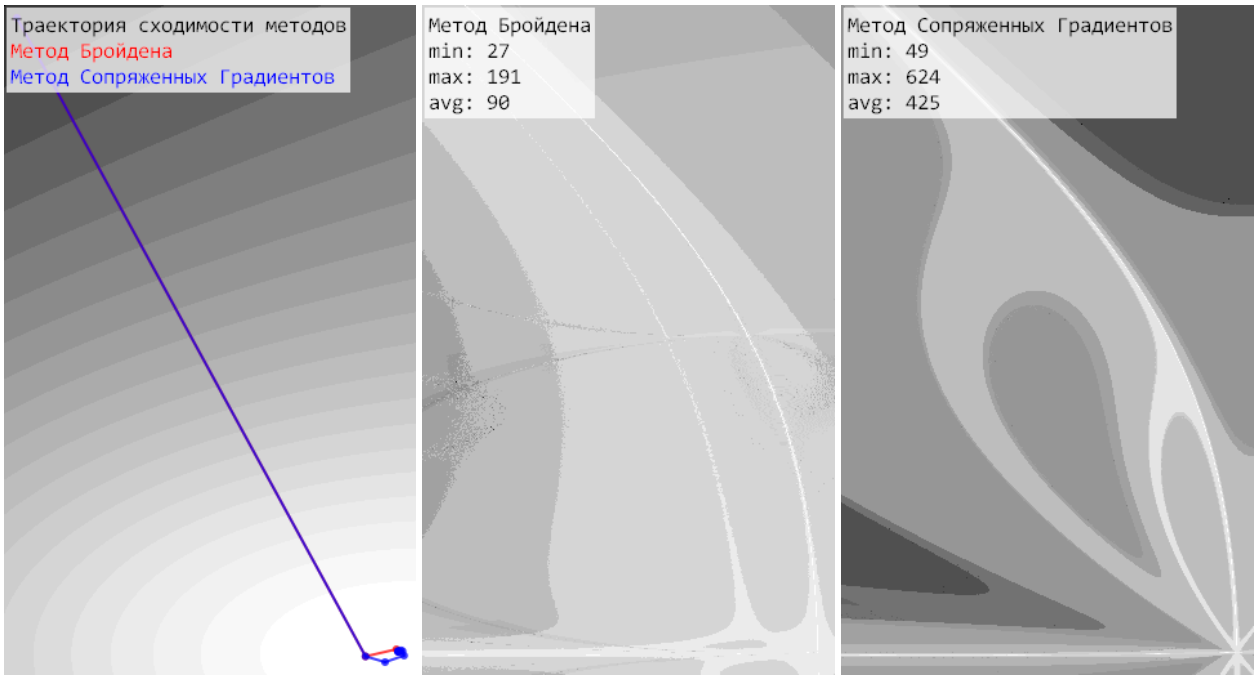
Метод Бройдена

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	3	81	$2.11 \cdot 10^{-5}$	-2.66
-4	3	96	$9.24 \cdot 10^{-6}$	-2.66
-5	3	111	$8.19 \cdot 10^{-6}$	-2.66
-6	4	164	$8.19 \cdot 10^{-6}$	-2.66

Метод Сопряженных Градиентов

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	10	451	$1.04 \cdot 10^{-3}$	-2.66
-4	10	451	$1.04 \cdot 10^{-3}$	-2.66
-5	13	587	$2.71 \cdot 10^{-5}$	-2.66
-6	15	676	$4.44 \cdot 10^{-7}$	-2.66

3.3.2 Изображения



3.3.3 Вторая таблица для метода Бройдена

i	(x,y)	$f(x,y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x,y)$	$H(f)$
0	$(0,0)^T$	-0.85	$(0,0)^T$	0	$(0,0)^T$	0	$(-0.19,-0.34)^T$	$\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$
1	$(1.1,2)^T$	-2.66	$(-0.19,-0.34)^T$	5.85	$(1.1,2)^T$	1.8	$(-0.098,0.054)^T$	$\begin{pmatrix}2.4 & 2.2\\ 2.2 & 4.6\end{pmatrix}$
2	$(1.2,2)^T$	-2.66	$(-0.12,0.027)^T$	0.83	$(0.096,-0.022)^T$	$5.29 \cdot 10^{-3}$	$(-0.66 \cdot 10^{-2},-0.029)^T$	$\begin{pmatrix}1.9 & 0.89\\ 0.89 & 1.3\end{pmatrix}$
3	$(1.2,2)^T$	-2.66	$(-0.038,-0.042)^T$	0.17	$(0.64 \cdot 10^{-2},0.71 \cdot 10^{-2})^T$	$1.24 \cdot 10^{-4}$	$(-0.67 \cdot 10^{-5},-0.2 \cdot 10^{-4})^T$	$\begin{pmatrix}1 & -0.016\\ -0.016 & 0.25\end{pmatrix}$

3.3.4 Вторая таблица для метода сопряженных градиентов

i	(x,y)	$f(x,y)$	s	λ	$\mathbf{x}_i - \mathbf{x}_{i-1}$	$f(\mathbf{x}_i) - f(\mathbf{x}_{i-1})$	$\nabla f(x,y)$	$H(f)$
0	$(0,0)^T$	-0.85	$(0,0)^T$	0	$(0,0)^T$	0	$(-0.19,-0.34)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
1	$(1.1,2)^T$	-2.66	$(0.19,0.34)^T$	5.85	$(1.1,2)^T$	1.8	$(-0.19,-0.34)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
2	$(1.1,2)^T$	-2.66	$(0.15,0.045)^T$	0.4	$(0.061,0.018)^T$	$2.51 \cdot 10^{-3}$	$(-0.098,0.054)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
3	$(1.2,2)^T$	-2.66	$(0.22,-0.076)^T$	0.27	$(0.059,-0.02)^T$	$2.43 \cdot 10^{-3}$	$(-0.038,0.13)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
4	$(1.2,2)^T$	-2.66	$(-0.018,-0.051)^T$	0.27	$(-0.47 \cdot 10^{-2},-0.014)^T$	$3.94 \cdot 10^{-4}$	$(0.018,0.051)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
5	$(1.2,2)^T$	-2.66	$(-0.016,-0.8 \cdot 10^{-2})^T$	0.31	$(-0.51 \cdot 10^{-2},-0.25 \cdot 10^{-2})^T$	$2.57 \cdot 10^{-5}$	$(0.012,-0.41 \cdot 10^{-2})^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
6	$(1.2,2)^T$	-2.66	$(-0.028,0.45 \cdot 10^{-2})^T$	0.33	$(-0.92 \cdot 10^{-2},0.15 \cdot 10^{-2})^T$	$4.35 \cdot 10^{-5}$	$(0.71 \cdot 10^{-2},-0.014)^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
7	$(1.2,2)^T$	-2.66	$(0.15 \cdot 10^{-2},0.93 \cdot 10^{-2})^T$	0.25	$(0.38 \cdot 10^{-3},0.24 \cdot 10^{-2})^T$	$1.12 \cdot 10^{-5}$	$(-0.15 \cdot 10^{-2},-0.93 \cdot 10^{-2})^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
8	$(1.2,2)^T$	-2.66	$(0.11 \cdot 10^{-2},0.81 \cdot 10^{-3})^T$	0.24	$(0.27 \cdot 10^{-3},0.2 \cdot 10^{-3})^T$	$1.15 \cdot 10^{-7}$	$(-0.96 \cdot 10^{-3},0.16 \cdot 10^{-3})^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
9	$(1.2,2)^T$	-2.66	$(0.2 \cdot 10^{-2},0.2 \cdot 10^{-4})^T$	0.35	$(0.72 \cdot 10^{-3},0.7 \cdot 10^{-5})^T$	$2.44 \cdot 10^{-7}$	$(-0.69 \cdot 10^{-3},0.96 \cdot 10^{-3})^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$
10	$(1.2,2)^T$	-2.66	$(0.1 \cdot 10^{-4},-0.1 \cdot 10^{-2})^T$	0.25	$(0.26 \cdot 10^{-5},-0.26 \cdot 10^{-3})^T$	$1.35 \cdot 10^{-7}$	$(-0.1 \cdot 10^{-4},0.1 \cdot 10^{-2})^T$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$

3.4 Функция по варианту + метод парабол

Для одномерного поиска использовался метод поиска отрезка из прошлой лабы + метод парабол на этом отрезке.

3.4.1 Первая таблица

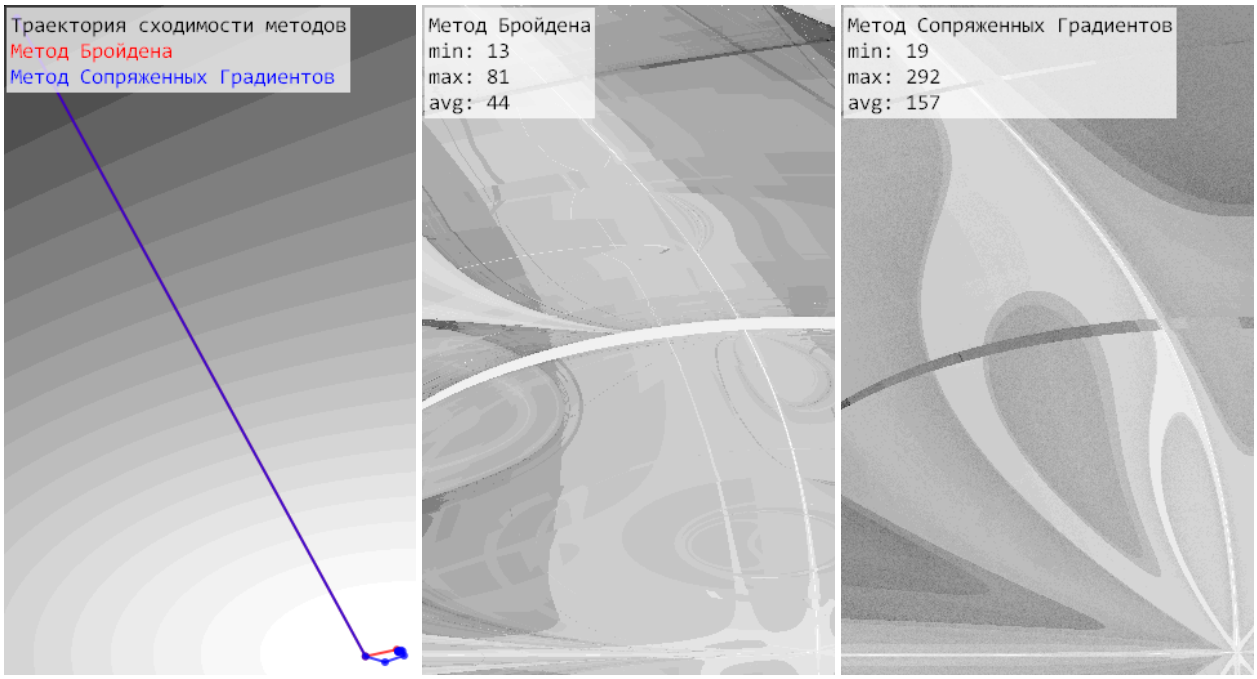
Метод Бройдена

$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	3	43	$3.97 \cdot 10^{-6}$	-2.66
-4	3	46	$6.6 \cdot 10^{-6}$	-2.66
-5	3	48	$8.19 \cdot 10^{-6}$	-2.66
-6	4	70	$8.88 \cdot 10^{-7}$	-2.66

Метод Сопряженных Градиентов

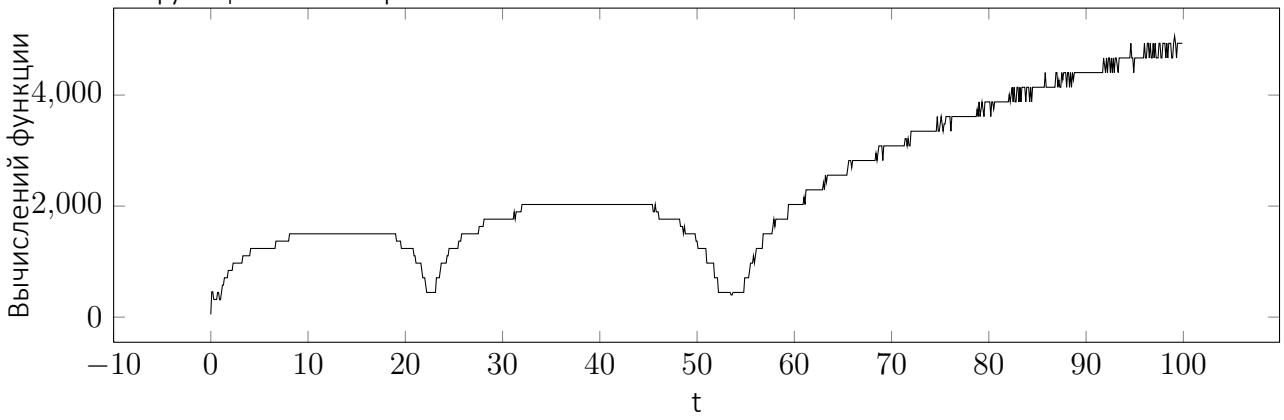
$\varepsilon = 10^i$	Итераций	Вычислений f	$\|\|\nabla f(x,y)\|\|_{L_2}$	$f(x,y)$
-3	10	157	$1.04 \cdot 10^{-3}$	-2.66
-4	10	157	$1.04 \cdot 10^{-3}$	-2.66
-5	13	228	$3.33 \cdot 10^{-5}$	-2.66
-6	16	310	$4 \cdot 10^{-6}$	-2.66

3.4.2 Изображения



3.5 Исследования МСГ на функции Розенброка

Функция $t \cdot (y - x^2)^2 + (1 - x)^2$. При $t = 0$ это квадратичная функция, при $t = 100$ эта функция становится функцией Розенброка.



4 Выводы

1. *О объеме вычислений в зависимости от точности:*
   - Для метода Бройдена количество итераций на всех трех функциях получается примерно одинаковым, увеличивается только число вычислений функции. Наверняка это связано с одно-

мерным поиском. Скорее всего число вычислений функции можно значительно уменьшить, задавая специальную точность для одномерного поиска: на первых итерациях маленькую точность; на последней итерации необходимую точность.

● Для МСГ количество итераций на всех трех функциях растет с ростом требуемой точности. Так же растет количество вычислений функции.

2. *О объеме вычислений в зависимости от начального приближения:*

● Для метода Бройдена на функции по варианту плотность светлых областей больше, чем для МСГ. Это означает, что вероятность выбрать хорошее начальное приближение для Метода Бройдена намного выше. Для метода же сопряженных градиентов много темных областей, и площадь светлых значительно меньше.

● Чем ближе к точке минимума, тем быстрее получается сходимость обоих методов. Но всё же существуют некоторые области, на расстоянии от минимума, на которые это правило не распространяется.

● Для обоих методов есть некоторая кривая, исходящая из минимума, на которой оба метода сходятся очень быстро. Пока не понятно что это за кривая.

3. *Сравнение метода парабол и золотого сечения:* метод парабол практически не влияет на число итераций, но зато значительно влияет на число вычислений функции. По сравнению с методом золотого сечения количество вычислений функции уменьшилось чуть больше, чем вдвое.
4. *О сходимости МСГ на функции розенброка:* на функции Розенброка МСГ сходится очень плохо. И изменение параметра от 0 до 100 незначительно влияет на это. Иногда наблюдаются локальные минимумы.
5. *Сравнение МСГ и метода Бройдена:* на данных функциях метод Бройдена значительно лучше МСГ.

5 Код программы

5.1 Заголовочные файлы

FILE **funcs.h**

```h
#pragma once

#include "methods.h"

double f1(const Vector& v);
double f2(const Vector& v);
double f3(const Vector& v);
```

FILE **methods.h**

```h
#pragma once

#include <iostream>
#include <vector>
#include <functional>
#include <Eigen/Dense>

#include <functional>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cmath>
#include <vector>

//---------------------------------------------------------------------------
// Функции для работы с одномерными функциями
typedef std::function<double(const double&)> OneDimensionFunction;
typedef std::function<double(const OneDimensionFunction&, double, double, double)>
↪  OneDimenshionExtremumFinder;
double optimizeDichotomy(const OneDimensionFunction& f, double a, double b, double eps);
double optimizeGoldenRatio(const OneDimensionFunction& f, double a, double b, double eps);
```

```cpp
double optimizeFibonacci(const OneDimensionFunction& f, double a, double b, double eps);
double optimizeParabola(const OneDimensionFunction& f, double a, double b, double eps);
void findSegment(const OneDimensionFunction& f, double x0, double& a, double& b, double eps = 1);

//-----------------------------------------------------------------------------
// Работа с многомерными функциями
struct MethodResult;

// Определяем используемые типы данных
typedef Eigen::MatrixXd Matrix;
typedef Eigen::VectorXd Vector;
typedef std::function<double(const Vector&)> Function;
typedef std::function<double(const OneDimensionFunction&, const double&)> ArgMinFunction;
typedef std::function<MethodResult(const Function&, const ArgMinFunction&, const Vector&, const
↪  double&)> Optimizator;

/** Доступная информация на каждом шаге. Нужна для построения таблиц и визуализации. */
struct StepInformation
{
    Vector point;   /// (x_i, y_i) - текущая точка
    double value;   /// f(x_i, y_i) - значение функции в этой точке
    Vector dir;     /// (s_1, s_2) - направление поиска
    double lambda;  /// Экстремум одномерного поиска
    Vector grad;    /// Градиент (для обоих методов необходимо)
    Matrix hessian; /// Матрица вторых производных, только для метода Бройдена
};

/** Определяет, по какой причине был совершен выход из функции. */
enum ExitType
{
    EXIT_RESIDUAL,
    EXIT_STEP,
    EXIT_ITERATIONS,
    EXIT_ERROR
};

/** Результат работы метода. */
struct MethodResult
{
    Vector answer;  /// Точка минимума
    int iterations; /// Число итераций
    ExitType exit;  /// Причина выхода
    int fCount;     /// Число вычислений функции

    // Информация о каждом шаге процесса оптимизации, необходимо для построения таблиц и визуализации
    std::vector<StepInformation> steps;
};

ArgMinFunction bindArgmin(const OneDimenshionExtremumFinder& finder);

Vector grad(const Function& f, const Vector& x);

/** Позволяет считать число вызовов функции. */
Function setFunctionToCountCalls(int* where, const Function& f);

/** Оптимизация с помощью метода Бройдена. */
MethodResult optimizeBroyden(const Function& f, const ArgMinFunction& argmin, const Vector& x0, const
↪  double& eps);

/** Оптимизация с помощью метода сопряженных градиентов Флетчера-Ривса. */
MethodResult optimizeConjugateGradient(const Function& f, const ArgMinFunction& argmin, const Vector&
↪  x0, const double& eps);

// Функции для вывода векторов и матриц.
std::string write_for_latex_double(double v, int precision);
std::ostream& operator<<(std::ostream& out, const Vector& v);
std::ostream& operator<<(std::ostream& out, const Matrix& m);

std::ostream& operator<<(std::ostream& out, const ExitType& e);
```

FILE **visualize.h**

```cpp
#pragma once

#include <vector>
#include <string>

#include "../methods.h"

void visualize(
    const Optimizator& o1,
    const Optimizator& o2,
    const ArgMinFunction& argmin,
    const Function& f,
    const Vector& x0,
    const double& eps,
```

```
    const int& size,
    const std::wstring& o1name,
    const std::wstring& o2name,
    const std::string& file
);
```

5.2 Файлы исходного кода

FILE **make_tables.cpp**

```
#include <iostream>
#include <iomanip>
#include <string>
#include <fstream>
#include "methods.h"
#include "funcs.h"
#include "visualize/visualize.h"

void makeFirstTable(const Optimizator& o, const ArgMinFunction& argmin, const Function& f, const Vector& x0, const std::string& file) {
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);
    fout << "x0 = " << x0 << std::endl;
    fout << "10^i\titer\tfCount\tgrad.norm()\tvalue" << std::endl;
    for (int i = 3; i < 7; ++i) {
        double eps = pow(10.0, -double(i));
        auto result = o(f, argmin, x0, eps);
        fout << -i << "\t" << result.iterations << "\t" << result.fCount << "\t" << result.steps.back().grad.norm() << "\t" << result.steps.back().value << std::endl;
    }
    fout.close();
}

void makeSecondTable(const Optimizator& o, const ArgMinFunction& argmin, const Function& f, const Vector& x0, const double& eps, const std::string& file) {
    auto result = o(f, argmin, x0, eps);
    std::ofstream fout(file + ".txt");
    fout << std::setprecision(10);

    fout << "x0 = " << x0 << ", eps=" << eps << std::endl;

    fout << "answer: " << result.answer << ", fCount: " << result.fCount << ", exit type: " << result.exit << ", iterations: " << result.iterations << std::endl;

    fout << "i\tpoint\tvalue\tdir\tlambda\tdiff_point\tdiff_value\tgrad\thessian" << std::endl;
    auto lastPoint = result.steps.front().point;
    auto lastValue = result.steps.front().value;
    int counter = 0;
    for (auto& i : result.steps) {
        Vector diff_point = Vector(i.point - lastPoint);
        double diff_value = std::fabs(i.value - lastValue);
        fout << counter << "\t" << i.point << "\t" << i.value << "\t" << i.dir << "\t" << i.lambda << "\t" << diff_point << "\t" << diff_value << "\t" << i.grad << "\t\\scalebox{.5}{" << i.hessian << "}" << std::endl;
        lastPoint = i.point;
        lastValue = i.value;
        counter++;
        if (counter % 25 == 0 && counter != 0) {
            fout.close();
            fout.open(file + "_" + std::to_string(counter / 25) + ".txt");
            fout << "next table part" << std::endl << "empty line for compability" << std::endl;
            fout << "i\tpoint\tvalue\tdir\tlambda\tdiff_point\tdiff_value\tgrad\thessian" << std::endl;
        }
    }

    fout.close();
}

void makeTableForF2(const Vector& x0, const ArgMinFunction& argmin, const std::string& file) {
    std::ofstream fout(file + ".txt");
    fout << "Table for function t*(y-x^2)^2 + (1-x)^2: " << std::endl;
    fout << "t\tfCount" << std::endl;

    for (int i = 0; i < 1000; i++) {
        double t = i / 10.0;
        auto f = [t](const Vector& v) -> double {
            const double& x = v(0);
            const double& y = v(1);
            return t * (y - x * x)*(y - x * x) + (1 - x)*(1 - x);
        };
        auto result = optimizeConjugateGradient(f, argmin, x0, 1e-3);
        fout << t << "\t" << result.fCount << std::endl;
```

```
    }

    fout.close();
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
int main() {
    Vector x0(2); x0 << 0, 0;
    double eps = 1e-3;

    auto argmin = bindArgmin(optimizeGoldenRatio);
    auto argminParabola = bindArgmin(optimizeParabola);

    std::vector<std::pair<Function, std::string>> funcs = {
        {f1, "f1"},
        {f2, "f2"},
        {f3, "f3"},
    };

    for (auto& i : funcs) {
        std::cout << "Draw " << i.second << std::endl;
        // Строим рисунки зависимости числа вычислений функции от положения
        visualize(optimizeBroyden, optimizeConjugateGradient, argmin, i.first, x0, 0.001, 500, L"Метод
        ↪  Бройдена", L"Метод Сопряженных Градиентов", "image_" + i.second);

        // Первая и вторая таблица для метода Бройдена
        makeFirstTable(optimizeBroyden, argmin, i.first, x0, "table1_" + i.second + "_broyden");
        makeSecondTable(optimizeBroyden, argmin, i.first, x0, eps, "table2_" + i.second +"_broyden");

        // Первая и вторая таблица для метода сопряженных градиентов
        makeFirstTable(optimizeConjugateGradient, argmin, i.first, x0, "table1_" + i.second +
        ↪  "_gradient");
        makeSecondTable(optimizeConjugateGradient, argmin, i.first, x0, eps, "table2_" + i.second
        ↪  +"_gradient");
    }

    std::cout << "Draw parabola" << std::endl;
    auto f = funcs.back().first;
    visualize(optimizeBroyden, optimizeConjugateGradient, argminParabola, f, x0, 0.001, 500, L"Метод
    ↪  Бройдена", L"Метод Сопряженных Градиентов", "image_parabola");
    makeFirstTable(optimizeBroyden, argminParabola, f, x0, "table1_parabola_broyden");
    makeFirstTable(optimizeConjugateGradient, argminParabola, f, x0, "table1_parabola_gradient");

    makeTableForF2(x0, argmin, "table_f2");
}
```

FILE **funcs.cpp**

```
#include "funcs.h"

//-----------------------------------------------------------------------------
double f1(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    return 100 * (y - x)*(y - x) + (1 - x)*(1 - x);
}

//-----------------------------------------------------------------------------
double f2(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    return 100 * (y - x * x)*(y - x * x) + (1 - x)*(1 - x);
}

//-----------------------------------------------------------------------------
double f3(const Vector& v) {
    const double& x = v(0);
    const double& y = v(1);
    const double A1 = 2, A2 = 1, a1 = 1, a2 = 3, b1 = 2, b2 = 3, c1 = 2, c2 = 1, d1 = 1, d2 = 3;

    #define sqr(x) ((x)*(x))
    return -(A1/(1 + sqr((x-a1)/b1) + sqr((y-c1)/d1)) + A2/(1 + sqr((x-a2)/b2) + sqr((y-c2)/d2)));
    #undef sqr
}
```

FILE **methods.cpp**

```
//#include "pch.h"
#include "methods.h"

//-----------------------------------------------------------------------------
```

```cpp
double optimizeDichotomy(const OneDimensionFunction& f, double a, double b, double eps) {
    double beta = eps*0.9;
    double x1, x2;
    while (std::fabs(a-b) > eps) {
        x1 = (a + b - beta) / 2.0;
        x2 = (a + b + beta) / 2.0;

        if (f(x1) > f(x2)) {
            a = x1;
        } else {
            b = x2;
        }
    }

    return (a + b) / 2.0;
}

//-----------------------------------------------------------------------------
double optimizeGoldenRatio(const OneDimensionFunction& f, double a, double b, double eps) {
    double beta = eps * 0.9;

    const double GOLD_A((3 - sqrt(5.0)) / 2);
    const double GOLD_B((sqrt(5.0) - 1) / 2);

    double x1 = a + GOLD_A * (b - a);
    double x2 = a + GOLD_B * (b - a);

    double f1 = f(x1);
    double f2 = f(x2);

    while (std::fabs(a - b) > eps) {
        if (f1 > f2) {
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + GOLD_B * (b - a);
            f2 = f(x2);
        }
        else {
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + GOLD_A * (b - a);
            f1 = f(x1);
        }
    }

    return (a + b) / 2.0;
}

//-----------------------------------------------------------------------------
double optimizeFibonacci(const OneDimensionFunction& f, double a, double b, double eps) {
    // Вычисление через сумму целых чисел
    auto fibonacciN = [](const int n) -> double {
        double f0 = 1;
        double f1 = 1;
        for (int i = 2; i <= n; i++) {
            double temp = f1;
            f1 = f0 + f1;
            f0 = temp;
        }
        return f1;
    };

    // Ищем количество итераций
    int n = 0;
    for (; fibonacciN(n + 2) < (b - a) / eps; n++);

    const double fibN2 = fibonacciN(n + 2);

    const double length = b - a;
    double x1 = a + fibonacciN(n) / fibN2 * length;
    double x2 = a + b - x1;

    double f1 = f(x1);
    double f2 = f(x2);

    for(int i = 2; i < n+2; i++) {
        if (f1 > f2) {
            a = x1;
            x1 = x2;
            f1 = f2;

            x2 = a + (fibonacciN(n - i + 2) / fibN2) * length;
            f2 = f(x2);
        } else {
            b = x2;
            x2 = x1;
            f2 = f1;
```

```
            x1 = a + (fibonacciN(n - i + 1) / fibN2) * length;
            f1 = f(x1);
        }
    }

    return (a + b) / 2.0;
}

//-----------------------------------------------------------------------------
double optimizeParabola(const OneDimensionFunction& f, double a, double b, double eps) {
    /*кроме унимодальности, налагается дополнительное требование
    достаточной гладкости (по крайней мере, непрерывности).
    // x1 < x2 < x3
    // f(x1) >= f(x2)
    // f(x3) >= f(x2)
    */

    double x1 = a;
    double x2 = (a + b) / 2;
    double x3 = b;
    double x = 0;            // current approximate xmin
    double xPrev = 0;        // pervious approximate xmin

    double f1 = f(x1);
    double f2 = f(x2);
    double f3 = f(x3);
    double fa = f1, fb = f3;

    bool finding = true;
    int iter = 0;
    double fx;
    while (finding) {

        // parabola y = a0 + a1(x - x1) + a2(x - x1)(x - x2);
        double a0 = f1;
        double a1 = (f2 - f1) / (x2 - x1);
        double a2 = ((f3 - f1) / (x3 - x1) - (f2 - f1) / (x2 - x1)) / (x3 - x2);

        x = 0.5 * (x1 + x2 - a1 / a2); // next approximation of min point

        // check out
        if (iter++ > 0) {
            if (fabs(x - xPrev) < eps) {
                finding = false;
                break;
            }
        }

        fx = f(x);

        // find new section:
        xPrev = x;

        if (x <= x2) {
            if (fx >= f2) {
                // xmin  in [x, x3]
                x1 = x; f1 = fx;
                // x2 and x3 stay the same
            }
            else {    // fx < f2
                // xmin in [x1, x2]
                // x1 stay the same
                x3 = x2; f3 = f2;
                x2 = x; f2 = fx;
            }
        }
        else { // x > x2
            if (fx > f2) {
                // xmin in [x1, x]
                // x1 and x2 stay the same
                x3 = x, f3 = fx;
            }
            else {// fx <= f2
                // xmin in [x2, x3]
                // x3 stay the same
                x1 = x2; f1 = f2;
                x2 = x; f2 = fx;
            }
        }
    }

    if (fa < fx) {
        if (fb < fa)
            return b;
        else
            return a;
    }
    if (fb < fx) {
```

```
        if (fa < fb)
            return a;
        else
            return b;
    }

    return x;
}

//-----------------------------------------------------------------------------
void findSegment(const OneDimensionFunction& f, double x0, double& a, double& b, double eps) {
    double h;
    double f1 = f(x0);
    double f2 = f(x0 + 1e-9);

    if (f1 > f2) h = eps;
    else h = -eps;

    double x1 = x0, x2 = x0, xPrevious = x0;
    int i = 0;
    do {
        xPrevious = x1;
        f1 = f2;
        x1 = x2;
        h *= 2;
        x2 = x1 + h;
        f2 = f(x2);

        if (h > 512) break; // Костыль, потому что бывают не унимодальные функции
    } while (f2 < f1);

    if(h > 0) {
        a = xPrevious;
        b = x2;
    }
    else {
        a = x2;
        b = xPrevious;
    }

    // Дополнение к костылю
    if (h > 512) {
        if (h > 0) {
            a = x0;
            b = x2;
        } else {
            a = x2;
            b = x0;
        }
    }
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
ArgMinFunction bindArgmin(const OneDimenshionExtremumFinder& finder) {
    return [finder](const OneDimensionFunction& f, double eps) {
        double a, b;
        findSegment(f, 0, a, b);
        return finder(f, a, b, eps);
    };
}

//-----------------------------------------------------------------------------
Vector grad(const Function& f, const Vector& x1) {
    const double eps = 1e-9;
    const double fx1 = f(x1);

    Vector result(x1.size());
    Vector x = x1;

    for (int i = 0; i < x.size(); ++i) {
        x(i) += eps;
        result[i] = (f(x) - fx1) / eps;
        x(i) -= eps;
    }
    return result;
}

//-----------------------------------------------------------------------------
Function setFunctionToCountCalls(int* where, const Function& f) {
    (*where) = 0;
    return [where, f](const Vector& v) -> double {
        (*where)++;
        return f(v);
    };
}
```

```cpp

//-----------------------------------------------------------------------------
MethodResult optimizeBroyden(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const double& eps) {
    MethodResult result;
    auto f = setFunctionToCountCalls(&result.fCount, f1);
    result.iterations = 0;

    #ifdef _DEBUG
    #define debug(a) std::cout << #a << ": " << (a) << std::endl;
    #else
    #define debug(a) ;
    #endif

    Vector x = x0; debug(x);
    Vector lastx = x;

    Matrix n;
    n = Matrix::Identity(x.size(), x.size()); debug(n);
    Vector gradf = grad(f, x);  debug(gradf);

    Vector zero(2); zero << 0, 0;
    result.steps.push_back({ x, f1(x), {zero}, 0, {gradf}, {n} });
    while (true) {
        debug(gradf.norm());
        if (gradf.norm() < eps) {
            result.exit = ExitType::EXIT_RESIDUAL;
            break;
        }

        Vector ngradf = n * gradf; debug(ngradf);
        auto optimizeFunc = [f, ngradf, x] (double lambda) -> double {
            return f(x - lambda * ngradf);
        };
        double lambda = argmin(optimizeFunc, eps); debug(lambda);

        Vector dx = -lambda * ngradf; debug(dx);
        x = x + dx; debug(x);
        Vector old_gradf = gradf;
        gradf = grad(f, x); debug(gradf);
        Vector dg = gradf - old_gradf; debug(dg);
        Vector temp = dx - n * dg; debug(temp);
        Matrix dn = temp * temp.transpose() / (temp.transpose() * dg); debug(dn);
        n = n + dn; debug(n);
        result.iterations++;

        result.steps.push_back({ x, f1(x), ngradf, lambda, gradf, n });

        if ((x - lastx).norm() < eps) {
            result.exit = ExitType::EXIT_STEP;
            break;
        }
        /*if (result.iterations > 200) {
            result.exit = ExitType::EXIT_ITERATIONS;
            break;
        }*/

        lastx = x;
    }

    result.answer = x;

    return result;
}

//-----------------------------------------------------------------------------
MethodResult optimizeConjugateGradient(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const double& eps) {
    MethodResult result;
    auto f = setFunctionToCountCalls(&result.fCount, f1);
    result.iterations = 0;

    double argmineps = 1e-7;
    double gradNorm = 0, grad1Norm = 0;

    ////////////////////////////////////////////////////
#ifdef _DEBUG
#define debug(a) std::cout << #a << ": " << (a) << std::endl;
#else
#define debug(a) ;
#endif

    // prepare calculation:
    int dim = x0.size();

    Vector x = x0, x1(dim);
    Vector s0(dim), s1(dim);
    Vector grad0(dim), grad1(dim);
```

```
    Vector zero = Vector::Zero(x0.size());
    Matrix zeroM = Matrix::Zero(x0.size(), x0.size());
    result.steps.push_back({ x, f1(x), zero, 0, grad(f, x), zeroM });
    bool optimization = true;
    while (optimization) {
        // find first direction
        grad0 = grad(f, x); //debug(grad0);
        s0 = -grad0;        //debug(s0);

        gradNorm = s0.norm();
        if (gradNorm < eps) {
            result.steps.push_back({ x1, f1(x1), s0, 0, grad0, zeroM });
            optimization = false;
            result.exit = ExitType::EXIT_STEP;
            break;
        }

        // processing K step ( k = 1,2,...,dim )
        for (int i = 0; i < dim+1; i++) {
            // calculate x(k)
        //    debug(s0);
        //    debug(x);
            auto optimizeFunc = [f, s0, x](double lambda) -> double {
                return f(x + lambda * s0);
            };

            double lambda = argmin(optimizeFunc, argmineps); //debug(lambda);
            x1 = x + lambda * s0; //debug(x1);

            result.steps.push_back({ x1, f1(x1), s0, lambda, grad0, zeroM });
            result.iterations++;

            // calculate direction to x(k+1)
            grad1 = grad(f, x1);                    //debug(grad1);
            grad1Norm = grad1.norm();          //debug(grad1Norm);
            double w = grad1Norm / gradNorm; //debug(w);
            s1 = -grad1 + w * s0;                    //debug(s1);

            // prepare next iteration:
            std::swap(s0, s1);
            std::swap(grad0, grad1);
            std::swap(x, x1);
            std::swap(gradNorm, grad1Norm);

            // check exit
            //double norm = s0.norm();                debug(norm);
            if (s0.norm() < eps) {
                optimization = false;
                result.exit = ExitType::EXIT_STEP;
                break;
            }

            debug(i);
        }
    }


    // k == n
    // swap x0 and x.


    ////////////////////////////////////////////
    result.answer = x;

    return result;
}

//---------------------------------------------------------------------------
std::string write_for_latex_double(double v, int precision) {
    int power = log(std::fabs(v)) / log(10.0);
    double value = v / pow(10.0, power);

    if (v == 0) {
        power = 0;
        value = 0;
    }

    std::stringstream sout;
    sout.precision(2);
    if (power == -1 || power == 0 || power == 1) {
        sout << v;
    } else {
        sout << value << "\\cdot 10^{" << power << "}";
    }

    return sout.str();
}
```

```
//---------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const Vector& v) {
    auto old = out.precision(); out.precision(2);
    out << "$(";
    for (int i = 0; i < v.size(); ++i) {
        out << write_for_latex_double(v(i), 2);
        if (i != v.size()-1)
            out << ", ";
    }
    out << ")^T$";
    out.precision(old);
    return out;
}

//---------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const Matrix& m) {
    auto old = out.precision(); out.precision(2);
    out << "$\\left(\\,\\begin{matrix}";
    for (int i = 0; i < m.rows(); ++i) {
        for (int j = 0; j < m.cols(); ++j) {
            out << write_for_latex_double(m(i, j), 2);
            if (j != m.cols()-1)
                out << "&";
        }
        if (i != m.rows()-1)
            out << "\\\\";
    }
    out << "\\end{matrix}\\,\\right)$";
    out.precision(old);
    return out;
}

//---------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const ExitType& e) {
    switch (e) {
        case EXIT_RESIDUAL: out << "by residual"; break;
        case EXIT_STEP: out << "by step"; break;
        case EXIT_ITERATIONS: out << "by iterations"; break;
        case EXIT_ERROR: out << "by error"; break;
    }
    return out;
}
```

FILE **visualize.cpp**

```
//#include "pch.h"
#include "methods.h"

//---------------------------------------------------------------------------
double optimizeDichotomy(const OneDimensionFunction& f, double a, double b, double eps) {
    double beta = eps*0.9;
    double x1, x2;
    while (std::fabs(a-b) > eps) {
        x1 = (a + b - beta) / 2.0;
        x2 = (a + b + beta) / 2.0;

        if (f(x1) > f(x2)) {
            a = x1;
        } else {
            b = x2;
        }
    }

    return (a + b) / 2.0;
}

//---------------------------------------------------------------------------
double optimizeGoldenRatio(const OneDimensionFunction& f, double a, double b, double eps) {
    double beta = eps * 0.9;

    const double GOLD_A((3 - sqrt(5.0)) / 2);
    const double GOLD_B((sqrt(5.0) - 1) / 2);

    double x1 = a + GOLD_A * (b - a);
    double x2 = a + GOLD_B * (b - a);

    double f1 = f(x1);
    double f2 = f(x2);

    while (std::fabs(a - b) > eps) {
        if (f1 > f2) {
            a = x1;
            x1 = x2;
            f1 = f2;
            x2 = a + GOLD_B * (b - a);
            f2 = f(x2);
```

```
        }
        else {
            b = x2;
            x2 = x1;
            f2 = f1;
            x1 = a + GOLD_A * (b - a);
            f1 = f(x1);
        }
    }

    return (a + b) / 2.0;
}

//-----------------------------------------------------------------------------
double optimizeFibonacci(const OneDimensionFunction& f, double a, double b, double eps) {
    // Вычисление через сумму целых чисел
    auto fibonacciN = [](const int n) -> double {
        double f0 = 1;
        double f1 = 1;
        for (int i = 2; i <= n; i++) {
            double temp = f1;
            f1 = f0 + f1;
            f0 = temp;
        }
        return f1;
    };

    // Ищем количество итераций
    int n = 0;
    for (; fibonacciN(n + 2) < (b - a) / eps; n++);

    const double fibN2 = fibonacciN(n + 2);

    const double length = b - a;
    double x1 = a + fibonacciN(n) / fibN2 * length;
    double x2 = a + b - x1;

    double f1 = f(x1);
    double f2 = f(x2);

    for(int i = 2; i < n+2; i++) {
        if (f1 > f2) {
            a = x1;
            x1 = x2;
            f1 = f2;

            x2 = a + (fibonacciN(n - i + 2) / fibN2) * length;
            f2 = f(x2);
        } else {
            b = x2;
            x2 = x1;
            f2 = f1;

            x1 = a + (fibonacciN(n - i + 1) / fibN2) * length;
            f1 = f(x1);
        }
    }

    return (a + b) / 2.0;
}

//-----------------------------------------------------------------------------
double optimizeParabola(const OneDimensionFunction& f, double a, double b, double eps) {
    /*кроме унимодальности, налагается дополнительное требование
    достаточной гладкости (по крайней мере, непрерывности).
    // x1 < x2 < x3
    // f(x1) >= f(x2)
    // f(x3) >= f(x2)
    */

    double x1 = a;
    double x2 = (a + b) / 2;
    double x3 = b;
    double x = 0;            // current approximate xmin
    double xPrev = 0;        // pervious approximate xmin

    double f1 = f(x1);
    double f2 = f(x2);
    double f3 = f(x3);
    double fa = f1, fb = f3;

    bool finding = true;
    int iter = 0;
    double fx;
    while (finding) {

        // parabola y = a0 + a1(x - x1) + a2(x - x1)(x - x2);
        double a0 = f1;
        double a1 = (f2 - f1) / (x2 - x1);
```

```
        double a2 = ((f3 - f1) / (x3 - x1) - (f2 - f1) / (x2 - x1)) / (x3 - x2);

        x = 0.5 * (x1 + x2 - a1 / a2); // next approximation of min point

        // check out
        if (iter++ > 0) {
            if (fabs(x - xPrev) < eps) {
                finding = false;
                break;
            }
        }

        fx = f(x);

        // find new section:
        xPrev = x;

        if (x <= x2) {
            if (fx >= f2) {
                // xmin  in [x, x3]
                x1 = x; f1 = fx;
                // x2 and x3 stay the same
            }
            else {    // fx < f2
                // xmin in [x1, x2]
                // x1 stay the same
                x3 = x2; f3 = f2;
                x2 = x; f2 = fx;
            }
        }
        else { // x > x2
            if (fx > f2) {
                // xmin in [x1, x]
                // x1 and x2 stay the same
                x3 = x, f3 = fx;
            }
            else {// fx <= f2
                // xmin in [x2, x3]
                // x3 stay the same
                x1 = x2; f1 = f2;
                x2 = x; f2 = fx;
            }
        }
    }

    if (fa < fx) {
        if (fb < fa)
            return b;
        else
            return a;
    }
    if (fb < fx) {
        if (fa < fb)
            return a;
        else
            return b;
    }

    return x;
}

//-----------------------------------------------------------------------------
void findSegment(const OneDimensionFunction& f, double x0, double& a, double& b, double eps) {
    double h;
    double f1 = f(x0);
    double f2 = f(x0 + 1e-9);

    if (f1 > f2) h = eps;
    else h = -eps;

    double x1 = x0, x2 = x0, xPrevious = x0;
    int i = 0;
    do {
        xPrevious = x1;
        f1 = f2;
        x1 = x2;
        h *= 2;
        x2 = x1 + h;
        f2 = f(x2);

        if (h > 512) break; // Костыль, потому что бывают не унимодальные функции
    } while (f2 < f1);

    if(h > 0) {
        a = xPrevious;
        b = x2;
    }
    else {
        a = x2;
```

```
        b = xPrevious;
    }

    // Дополнение к костылю
    if (h > 512) {
        if (h > 0) {
            a = x0;
            b = x2;
        } else {
            a = x2;
            b = x0;
        }
    }
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
ArgMinFunction bindArgmin(const OneDimenshionExtremumFinder& finder) {
    return [finder](const OneDimensionFunction& f, double eps) {
        double a, b;
        findSegment(f, 0, a, b);
        return finder(f, a, b, eps);
    };
}

//-----------------------------------------------------------------------------
Vector grad(const Function& f, const Vector& x1) {
    const double eps = 1e-9;
    const double fx1 = f(x1);

    Vector result(x1.size());
    Vector x = x1;

    for (int i = 0; i < x.size(); ++i) {
        x(i) += eps;
        result[i] = (f(x) - fx1) / eps;
        x(i) -= eps;
    }
    return result;
}

//-----------------------------------------------------------------------------
Function setFunctionToCountCalls(int* where, const Function& f) {
    (*where) = 0;
    return [where, f](const Vector& v) -> double {
        (*where)++;
        return f(v);
    };
}

//-----------------------------------------------------------------------------
MethodResult optimizeBroyden(const Function& f1, const ArgMinFunction& argmin, const Vector& x0, const
↪   double& eps) {
    MethodResult result;
    auto f = setFunctionToCountCalls(&result.fCount, f1);
    result.iterations = 0;

    #ifdef _DEBUG
    #define debug(a) std::cout << #a << ": " << (a) << std::endl;
    #else
    #define debug(a) ;
    #endif

    Vector x = x0; debug(x);
    Vector lastx = x;

    Matrix n;
    n = Matrix::Identity(x.size(), x.size()); debug(n);
    Vector gradf = grad(f, x);  debug(gradf);

    Vector zero(2); zero << 0, 0;
    result.steps.push_back({ x, f1(x), {zero}, 0, {gradf}, {n} });
    while (true) {
        debug(gradf.norm());
        if (gradf.norm() < eps) {
            result.exit = ExitType::EXIT_RESIDUAL;
            break;
        }

        Vector ngradf = n * gradf; debug(ngradf);
        auto optimizeFunc = [f, ngradf, x] (double lambda) -> double {
            return f(x - lambda * ngradf);
        };
        double lambda = argmin(optimizeFunc, eps); debug(lambda);

        Vector dx = -lambda * ngradf; debug(dx);
```

```
        x = x + dx; debug(x);
        Vector old_gradf = gradf;
        gradf = grad(f, x); debug(gradf);
        Vector dg = gradf - old_gradf; debug(dg);
        Vector temp = dx - n * dg; debug(temp);
        Matrix dn = temp * temp.transpose() / (temp.transpose() * dg); debug(dn);
        n = n + dn; debug(n);
        result.iterations++;

        result.steps.push_back({ x, f1(x), ngradf, lambda, gradf, n });

        if ((x - lastx).norm() < eps) {
            result.exit = ExitType::EXIT_STEP;
            break;
        }
        /*if (result.iterations > 200) {
            result.exit = ExitType::EXIT_ITERATIONS;
            break;
        }*/

        lastx = x;
    }

    result.answer = x;

    return result;
}

//-----------------------------------------------------------------------------
MethodResult optimizeConjugateGradient(const Function& f1, const ArgMinFunction& argmin, const Vector&
↪  x0, const double& eps) {
    MethodResult result;
    auto f = setFunctionToCountCalls(&result.fCount, f1);
    result.iterations = 0;

    double argmineps = 1e-7;
    double gradNorm = 0, grad1Norm = 0;

    ////////////////////////////////////////////////////
#ifdef _DEBUG
#define debug(a) std::cout << #a << ": " << (a) << std::endl;
#else
#define debug(a) ;
#endif

    // prepare calculation:
    int dim = x0.size();

    Vector x = x0, x1(dim);
    Vector s0(dim), s1(dim);
    Vector grad0(dim), grad1(dim);

    Vector zero = Vector::Zero(x0.size());
    Matrix zeroM = Matrix::Zero(x0.size(), x0.size());
    result.steps.push_back({ x, f1(x), zero, 0, grad(f, x), zeroM });
    bool optimization = true;
    while (optimization) {
        // find first direction
        grad0 = grad(f, x); //debug(grad0);
        s0 = -grad0;        //debug(s0);

        gradNorm = s0.norm();
        if (gradNorm < eps) {
            result.steps.push_back({ x1, f1(x1), s0, 0, grad0, zeroM });
            optimization = false;
            result.exit = ExitType::EXIT_STEP;
            break;
        }

        // processing K step ( k = 1,2,...,dim )
        for (int i = 0; i < dim+1; i++) {
            // calculate x(k)
        //    debug(s0);
        //    debug(x);
            auto optimizeFunc = [f, s0, x](double lambda) -> double {
                return f(x + lambda * s0);
            };

            double lambda = argmin(optimizeFunc, argmineps); //debug(lambda);
            x1 = x + lambda * s0; //debug(x1);

            result.steps.push_back({ x1, f1(x1), s0, lambda, grad0, zeroM });
            result.iterations++;

            // calculate direction to x(k+1)
            grad1 = grad(f, x1);                    //debug(grad1);
            grad1Norm = grad1.norm();          //debug(grad1Norm);
            double w = grad1Norm / gradNorm; //debug(w);
```

```
            s1 = -grad1 + w * s0;                   //debug(s1);

            // prepare next iteration:
            std::swap(s0, s1);
            std::swap(grad0, grad1);
            std::swap(x, x1);
            std::swap(gradNorm, grad1Norm);

            // check exit
            //double norm = s0.norm();                debug(norm);
            if (s0.norm() < eps) {
                optimization = false;
                result.exit = ExitType::EXIT_STEP;
                break;
            }

            debug(i);
        }
    }


    // k == n
    // swap x0 and x.


    ////////////////////////////////////////////
    result.answer = x;

    return result;
}

//-----------------------------------------------------------------------------
std::string write_for_latex_double(double v, int precision) {
    int power = log(std::fabs(v)) / log(10.0);
    double value = v / pow(10.0, power);

    if (v == 0) {
        power = 0;
        value = 0;
    }

    std::stringstream sout;
    sout.precision(2);
    if (power == -1 || power == 0 || power == 1) {
        sout << v;
    } else {
        sout << value << "\\cdot 10^{" << power << "}";
    }

    return sout.str();
}

//-----------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const Vector& v) {
    auto old = out.precision(); out.precision(2);
    out << "$(";
    for (int i = 0; i < v.size(); ++i) {
        out << write_for_latex_double(v(i), 2);
        if (i != v.size()-1)
            out << ", ";
    }
    out << ")^T$";
    out.precision(old);
    return out;
}

//-----------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const Matrix& m) {
    auto old = out.precision(); out.precision(2);
    out << "$\\left(\\,\\begin{matrix}";
    for (int i = 0; i < m.rows(); ++i) {
        for (int j = 0; j < m.cols(); ++j) {
            out << write_for_latex_double(m(i, j), 2);
            if (j != m.cols()-1)
                out << "&";
        }
        if (i != m.rows()-1)
            out << "\\\\";
    }
    out << "\\end{matrix}\\,\\right)$";
    out.precision(old);
    return out;
}

//-----------------------------------------------------------------------------
std::ostream& operator<<(std::ostream& out, const ExitType& e) {
    switch (e) {
        case EXIT_RESIDUAL: out << "by residual"; break;
        case EXIT_STEP: out << "by step"; break;
```

```
484             case EXIT_ITERATIONS: out << ”by iterations”; break;
485             case EXIT_ERROR: out << ”by error”; break;
486         }
487         return out;
488 }
```