Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Новосибирский государственный технический университет»

Кафедра теоретической и прикладной информатики

Лабораторная работа №2
по дисциплине «Методы оптимизации»

Факультет:	ПМИ
Группа:	ПМ-92
Бригада:	7
Студенты:	Иванов В., Кутузов И.
Преподаватель:	Филиппова Е.В.

Новосибирск

2022

1. Цель работы

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

2. Задание

I. Реализовать два метода поиска экстремума функции (разного порядка). Включить в реализуемый алгоритм собственную процедуру, реализующую одномерный поиск по направлению. Выбранные методы должны иметь разный порядок.

II. Исследовать алгоритмы на функциях:

$$f(\overline{x}) = 100(x_2 - x_1)^2 + (1 - x_1)^2$$

$$f(\overline{x}) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2$$

$$f(x,\ y) = 2e^{-\left(\frac{x-1}{2}\right)^2 - \left(\frac{y-1}{2}\right)^2} + 3e^{-\left(\frac{x-1}{3}\right)^2 - \left(\frac{y-3}{3}\right)^2}$$

Спуск осуществлять из различных исходных точек (не менее двух). Исследовать сходимость алгоритма, фиксируя точность определения минимума/максимума, количество итераций метода и количество вычислений функции в зависимости от задаваемой точности поиска. Результатом выполнения данного пункта должны быть выводы об объеме вычислений в зависимости от задаваемой точности и начального приближения.

III. Построить траекторию спуска различных алгоритмов из одной и той же исходной точки с одинаковой точностью. В отчете наложить эту траекторию на рисунок с линиями равного уровня заданной функции.

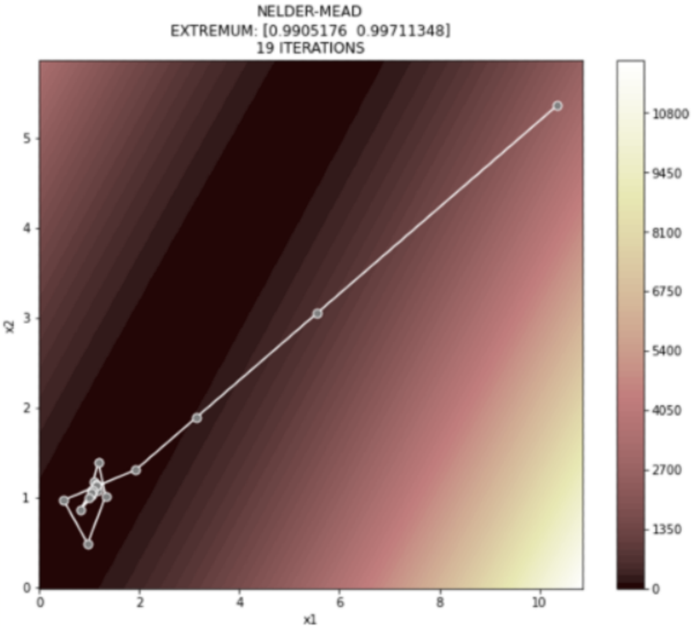
IV. Реализовать метод квадратичной интерполяции (метод парабол) для приближенного нахождения экстремума при одномерном поиске. Исследовать влияние точности одномерного поиска на общее количество итераций и вычислений функции при разных методах одномерного поиска.

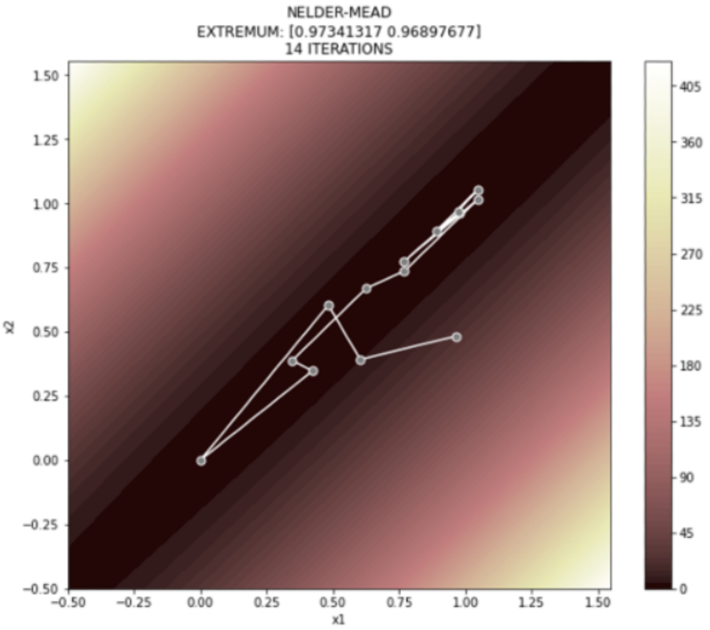
3. Исследование функций

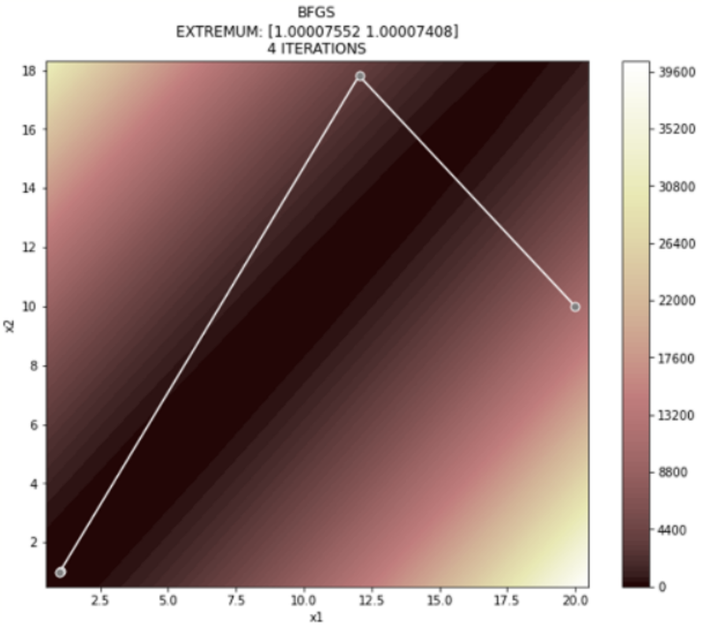
Функция 1

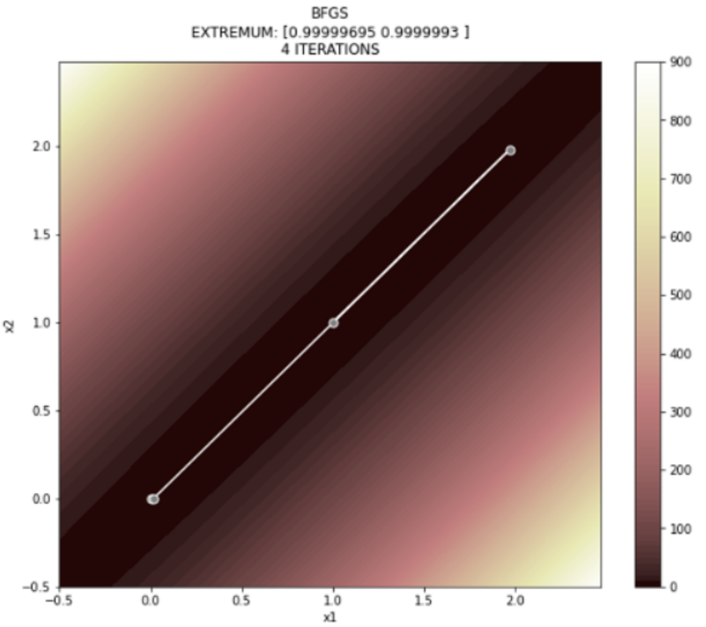
```
nelder_mead(f1, [20, 10], 1e-3)
nelder_mead(f1, [0, 0], 1e-3)
```

```
BFGS(f1, [20, 10], 1e-3)
BFGS(f1, [0, 0], 1e-3, table=True)
```







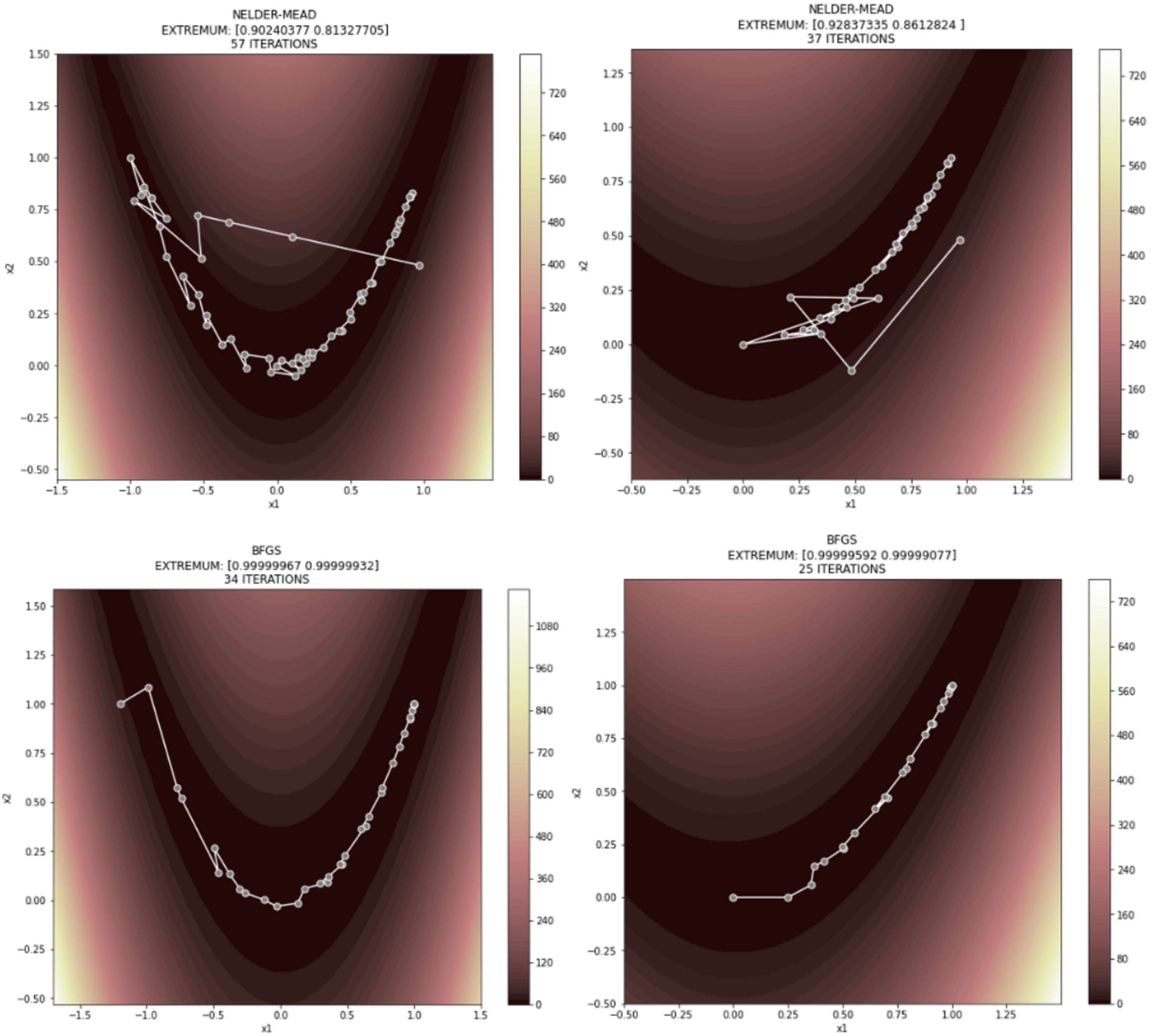


	x	y	f(x, y)	(s1, s2)	a	x_diff	y_diff	f_diff	angle	grad	hessian
1	0.000000	0.000000	1.000000e+00	0.016, 0.0	1.000000	0.000000	0.000000	1.000000	-	-2.0, 0.0	[1.0, 0.0, 0.0, 1.0]
2	0.015625	0.000000	9.934082e-01	1.955, 1.98	0.007812	0.015625	0.000000	0.006592	1.216416	1.156, -3.125	[1.0, 1.0, 1.0, 1.0]
3	1.970493	1.980198	9.512754e-01	-0.97, -0.98	1.000000	1.954868	1.980198	0.042133	0.782942	0.0, 1.941	[0.5, 0.5, 0.5, 0.5]
4	0.999997	0.999999	5.645503e-10	-	1.000000	0.970496	0.980199	0.951275	1.577232	-0.0, 0.0	[0.5, 0.5, 0.5, 0.5]

Функция 2

```
nelder_mead(f2, [-1, 1], 1e-3)
nelder_mead(f2, [0, 0], 1e-3)
```

```
BFGS(f2, [-1.2, 1], 1e-3)
BFGS(f2, [0, 0], 1e-3, table=True)
```

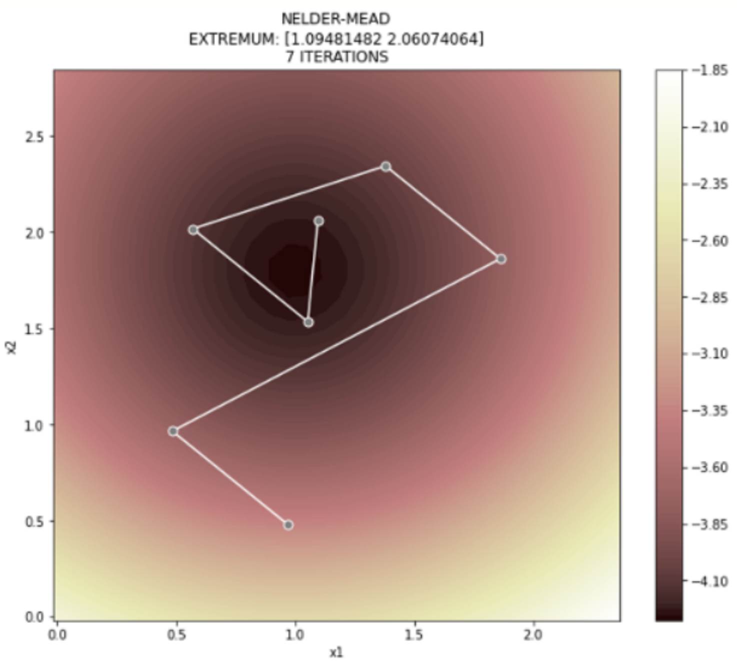


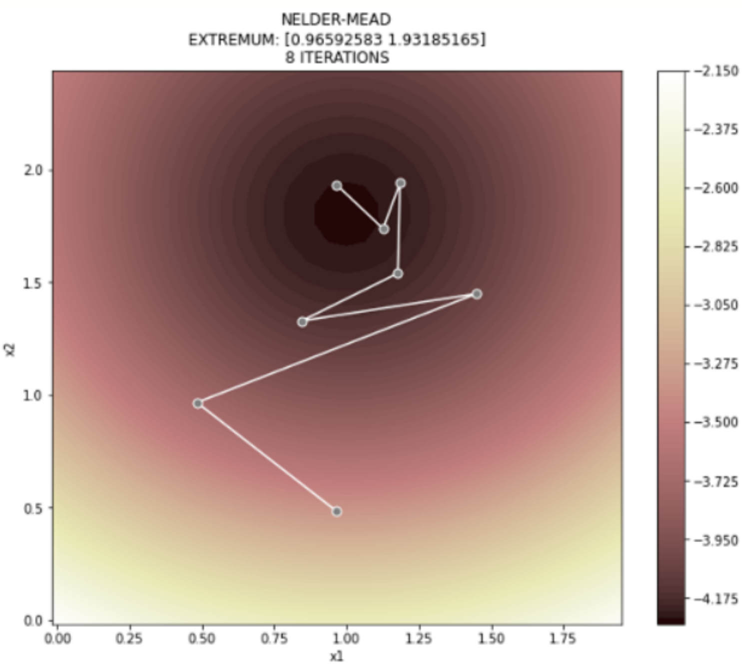
	x	y	f(x, y)	(s1, s2)	a	x_diff	y_diff	f_diff	angle	grad	hessian
1	0.000000	0.000000	1.000000e+00	0.25, 0.0	1.000000	0.000000	0.000000	1.000000e+00	-	-2.0, 0.0	[1.0, 0.0, 0.0, 1.0]
2	0.250000	0.000000	9.531250e-01	0.104, 0.058	0.125000	0.250000	0.000000	4.687500e-02	1.207649	4.75, -12.5	[3.5, 1.9, 1.9, 1.0]
3	0.354419	0.057870	8.756748e-01	0.014, 0.088	0.015625	0.104419	0.057870	7.745020e-02	1.182337	8.312, -13.548	[0.0, 0.0, 0.0, 0.0]
4	0.368495	0.145725	4.086716e-01	0.048, 0.025	0.500000	0.014077	0.087855	4.670032e-01	2.135355	-2.728, 1.987	[0.0, 0.0, 0.0, 0.0]
5	0.416141	0.170817	3.414464e-01	0.09, 0.062	1.000000	0.047646	0.025092	6.722524e-02	2.984974	-0.775, -0.471	[0.2, 0.1, 0.1, 0.1]
6	0.506440	0.232597	3.006495e-01	-0.007, 0.005	0.500000	0.090299	0.061780	4.079694e-02	1.322805	3.851, -4.777	[0.1, 0.1, 0.1, 0.0]
7	0.499637	0.237516	2.650566e-01	0.054, 0.065	1.000000	0.006803	0.004918	3.559286e-02	1.484123	1.422, -2.424	[0.1, 0.1, 0.1, 0.1]

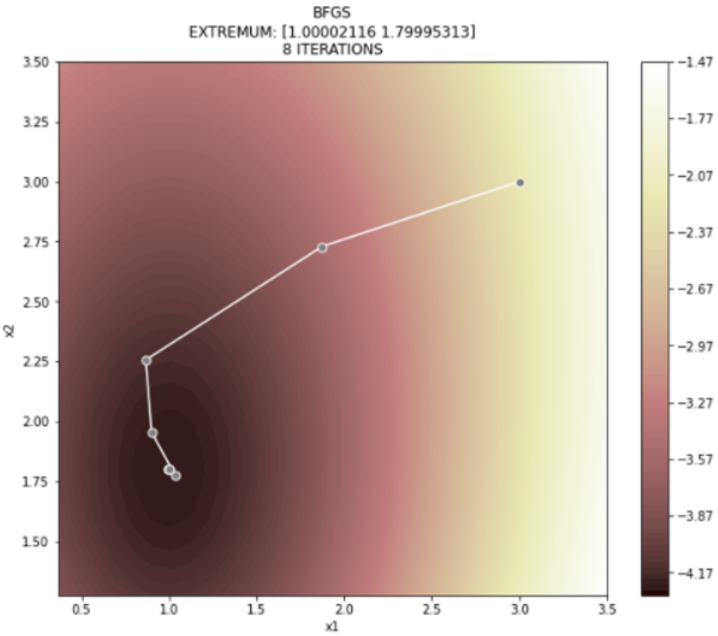
18	0.984266	0.959460	8.932330e-03	-0.038, -0.065	1.000000	0.079675	0.142292	2.950922e-04	1.246145	3.638, -1.864	[0.2, 0.4, 0.4, 0.7]
19	0.945904	0.894388	2.938350e-03	0.016, 0.03	1.000000	0.038362	0.065072	5.993980e-03	2.013271	0.023, -0.069	[0.1, 0.3, 0.3, 0.5]
20	0.961722	0.924226	1.511846e-03	0.033, 0.063	1.000000	0.015818	0.029838	1.426505e-03	1.398529	0.186, -0.137	[0.4, 0.9, 0.9, 1.6]
21	0.994725	0.987491	4.225820e-04	-0.003, -0.004	1.000000	0.033003	0.063265	1.089264e-03	1.252928	0.78, -0.397	[0.4, 0.7, 0.7, 1.4]
22	0.991915	0.983404	8.954701e-05	0.005, 0.011	1.000000	0.002810	0.004087	3.330350e-04	1.283648	0.179, -0.098	[0.3, 0.6, 0.6, 1.3]
23	0.997060	0.994083	8.853367e-06	0.003, 0.005	1.000000	0.005145	0.010680	8.069365e-05	1.420185	0.012, -0.009	[0.5, 0.9, 0.9, 1.8]
24	0.999749	0.999487	7.619242e-08	0.0, 0.001	1.000000	0.002689	0.005404	8.777175e-06	1.2966	0.004, -0.002	[0.5, 1.0, 1.0, 2.0]
25	0.999996	0.999991	1.321322e-10	-	1.000000	0.000247	0.000504	7.606029e-08	1.256739	0.0, -0.0	[0.5, 1.0, 1.0, 2.0]

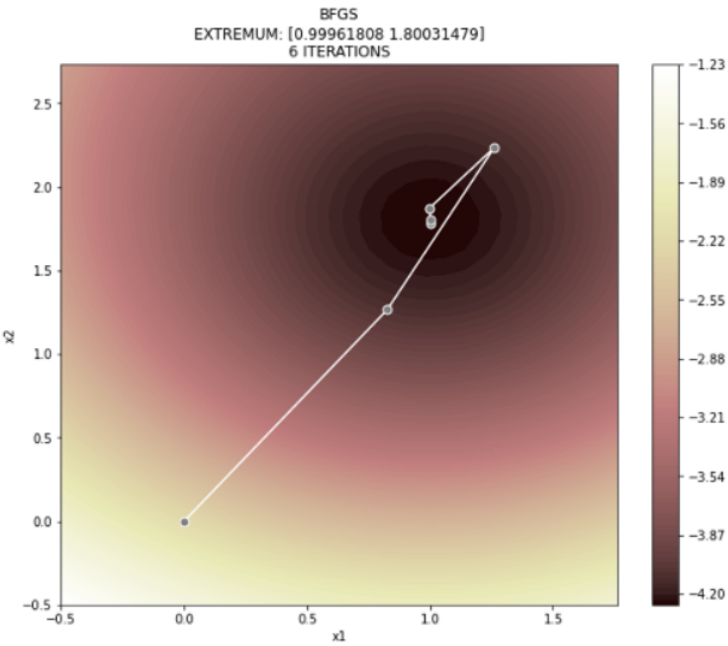
Функция 3

```
nelder_mead(f3, [3, 3], 1e-3)
nelder_mead(f3, [0, 0], 1e-3)
BFGS(f3, [3, 3], 1e-3)
BFGS(f3, [0, 0], 1e-3, table=True)
```









	x	y	f(x, y)	(s1, s2)	a	x_diff	y_diff	f_diff	angle	grad	hessian
1	0.000000	0.000000	-2.200640	0.826, 1.265	1	0.000000	0.000000	2.200640	-	-0.826, -1.265	[1.0, 0.0, 0.0, 1.0]
2	0.825993	1.264917	-4.090269	0.435, 0.968	1	0.825993	1.264917	1.889628	2.982153	-0.252, -0.567	[1.1, 0.3, 0.3, 1.6]
3	1.260521	2.233366	-4.133547	-0.264, -0.362	1	0.434529	0.968449	0.043279	0.246597	0.337, 0.354	[0.9, -0.1, -0.1, 1.1]
4	0.996818	1.870983	-4.258310	0.007, -0.089	1	0.263703	0.362383	0.124762	0.555963	-0.004, 0.067	[0.8, -0.1, -0.1, 1.3]
5	1.003971	1.781571	-4.260543	-0.004, 0.019	1	0.007153	0.089412	0.002234	2.322295	0.006, -0.018	[0.8, 0.0, 0.0, 1.1]
6	0.999618	1.800315	-4.260719	-	1	0.004352	0.018744	0.000176	1.566838	-0.001, 0.0	[0.8, 0.0, 0.0, 1.0]

4. Исследование сходимости

Далее в каждой из таблиц полагается начальная точка (0, 0).
Второй столбец – количество итераций, третий – количество вычислений функции, последний – значение функции в найденной точке.

Функция 1: Метод Бройдена

```
1e-03 4 59 (9.99997e-01, 9.99999e-01) 5.646e-10
1e-04 5 68 (1.00000e+00, 1.00000e+00) 3.924e-16
1e-05 5 68 (1.00000e+00, 1.00000e+00) 3.924e-16
1e-06 5 68 (1.00000e+00, 1.00000e+00) 3.924e-16
1e-07 6 77 (1.00000e+00, 1.00000e+00) 1.333e-28
```

Функция 1: Метод Нелдера-Мида

```
1e-03 14 68 (9.73413e-01, 9.68977e-01) 2.675e-03
1e-04 19 92 (1.00974e+00, 1.01026e+00) 1.217e-04
1e-05 20 97 (9.89745e-01, 9.89557e-01) 1.087e-04
1e-06 25 122 (9.98078e-01, 9.97971e-01) 4.840e-06
1e-07 29 142 (9.99480e-01, 9.99475e-01) 2.726e-07
```

Функция 2: Метод Бройдена

```
1e-03 25 268 (9.99996e-01, 9.99991e-01) 1.321e-10
1e-04 26 277 (1.00000e+00, 1.00000e+00) 2.367e-14
1e-05 26 277 (1.00000e+00, 1.00000e+00) 2.367e-14
1e-06 27 286 (1.00000e+00, 1.00000e+00) 4.500e-17
1e-07 27 286 (1.00000e+00, 1.00000e+00) 4.500e-17
```

Функция 2: Метод Нелдера-Мида

```
1e-03 37 155 (9.28373e-01, 8.61282e-01) 5.166e-03
1e-04 42 178 (9.91035e-01, 9.83475e-01) 2.557e-04
1e-05 46 197 (9.99648e-01, 9.99768e-01) 2.242e-05
1e-06 49 212 (9.98442e-01, 9.96816e-01) 2.921e-06
1e-07 54 235 (9.99687e-01, 9.99336e-01) 2.412e-07
```

Функция 3: Метод Бройдена

```
1e-03 6 49 (9.99618e-01, 1.80031e+00) -4.261e+00
1e-04 7 58 (1.00004e+00, 1.80002e+00) -4.261e+00
1e-05 8 67 (9.99999e-01, 1.80000e+00) -4.261e+00
1e-06 9 76 (1.00000e+00, 1.80000e+00) -4.261e+00
1e-07 9 76 (1.00000e+00, 1.80000e+00) -4.261e+00
```

Функция 3: Метод Нелдера-Мида

```
1e-03 8 38 (9.65926e-01, 1.93185e+00) -4.252e+00
1e-04 13 63 (9.76553e-01, 1.79326e+00) -4.260e+00
1e-05 17 82 (1.00634e+00, 1.79458e+00) -4.261e+00
1e-06 22 107 (1.00122e+00, 1.79887e+00) -4.261e+00
1e-07 24 117 (9.99350e-01, 1.80049e+00) -4.261e+00
```

5. Код программы

```python
def f1(x):
    return 100*(x[1] - x[0])**2 + (1 - x[0])**2

def f2(x):
    return 100*(x[1] - x[0]**2)**2 + (1 - x[0])**2

def f3(x):
    return - (2*np.exp(-((x[0]-1)/2)**2 - ((x[1]-1)/2)**2) +
             3*np.exp(-((x[0]-1)/3)**2 - ((x[1]-3)/3)**2))

def make_plot(x_store, f, algorithm):
    x1 = np.linspace(min(x_store[:, 0] - 0.5), max(x_store[:, 0] + 0.5), 30)
    x2 = np.linspace(min(x_store[:, 1] - 0.5), max(x_store[:, 1] + 0.5), 30)
    X1, X2 = np.meshgrid(x1, x2)
    Z = f([X1, X2])
    plt.figure(figsize=(10, 8))
    plt.title(algorithm + '\nEXTREMUM: ' + str(x_store[-1, :]) + '\n' + str(len(x_store)) + ' ITERATIONS')
    plt.contourf(X1, X2, Z, 100, cmap='pink')
    plt.colorbar()
    plt.scatter(x_store[:, 0], x_store[:, 1], c='grey', zorder=3, s=50, edgecolors='w')
    plt.plot(x_store[:, 0], x_store[:, 1], c='w')
    plt.xlabel('x1')
    plt.ylabel('x2')
```

```
    plt.show()

def make_table(x_store, f_store, nabla_store, a_store, s_store, H_store):
    angle = np.arccos((x_store[:, 0]*np.array(nabla_store)[:, 0] +
                       x_store[:, 1]*np.array(nabla_store)[:, 1]) /
                      (np.sqrt(x_store[:, 0]**2 + x_store[:, 1]**2) *
                       np.sqrt(np.array(nabla_store)[:, 0]**2 + np.array(nabla_store)[:, 1]**2)))
    nabla_store = [', '.join(item) for item in np.around(np.array(nabla_store), 3).astype(str)]
    s_store = [', '.join(item) for item in np.around(np.array(s_store), 3).astype(str)]
    x_diff = np.insert(np.absolute(x_store[1:, 0] - x_store[:-1, 0]), 0, 0.)
    y_diff = np.insert(np.absolute(x_store[1:, 1] - x_store[:-1, 1]), 0, 0.)
    f_store = np.array(f_store)
    f_diff = np.insert(np.absolute(f_store[1:] - f_store[:-1]), 0, np.absolute(f_store[0]))
    H_store = np.around(np.reshape(np.array(H_store), (1, len(H_store), 4))[0], 1)
    df = pd.DataFrame([x_store[:, 0], x_store[:, 1],
                       f_store, s_store, a_store,
                       x_diff, y_diff, f_diff,
                       angle, nabla_store, H_store]).T
    df.rename(columns={0: 'x', 1: 'y',
                       2: 'f(x, y)', 3: '(s1, s2)', 4: 'a',
                       5: 'x_diff', 6: 'y_diff', 7: 'f_diff',
                       8: 'angle', 9: 'grad', 10: 'hessian'}, inplace=True)
    df.fillna('-', inplace=True)
    df.index += 1
    return df
```

Метод Бройдена

```
f_count = 0

def BFGS(f, x0, eps, plot=True, table=False):
    global f_count
    f_store, nabla_store, a_store, s_store, H_store = [], [], [], [], []
    it, f_count = 1, 0
    x0 = list(map(float, x0))
    nabla = grad(f, x0)
    H = np.eye(2)
    x = x0[:]
    x_store =  np.zeros((1, 2))
    x_store[0, :] = x
    f_store.append(f(x))
    nabla_store.append(nabla)
    a_store.append(1)
    H_store.append(H)
    while np.linalg.norm(nabla) > eps:
        it += 1
        p = - H @ nabla
        a = line_search(f, x, p, nabla)
        s = a * p
        x_new = x + a * p
        nabla_new = grad(f, x_new)
        y = nabla_new - nabla
        y = np.array([y])
        s_store.append(s)
        s = np.array([s])
        y = np.reshape(y, (2, 1))
        s = np.reshape(s, (2, 1))
        r = 1 / (y.T @ s)
        li = (np.eye(2) - (r*((s @ (y.T)))))
        ri = (np.eye(2) - (r*((y @ (s.T)))))
        hess_inter = li @ H @ ri
        H = hess_inter + (r*((s @ (s.T))))
        nabla = nabla_new[:]
        x = x_new[:]
        x_store = np.append(x_store, [x], axis=0)
        f_store.append(f(x_store[-1]))
        nabla_store.append(nabla)
        a_store.append(a)
```

```
        H_store.append(H)
    if plot == True:
        make_plot(x_store, f, 'BFGS')
    else:
        print('{:.0e}'.format(eps), it, f_count,
              '(' + str('{:.5e}'.format(x_store[-1][0])) + ', ' + str('{:.5e}'.format(x_store[-1][1])) + ')',
              '{:.3e}'.format(abs(f_store[-1])))
    if table == True:
        df = make_table(x_store, f_store, nabla_store, a_store, s_store, H_store)
        return df

def line_search(f, x, p, nabla):
    global f_count
    a = 1
    c1 = 1e-4
    c2 = 0.9
    fx = f(x)
    f_count += 1
    x_new = x + a * p
    nabla_new = grad(f, x_new)
    while f(x_new) >= fx + (c1*a*nabla.T @ p) or nabla_new.T @ p <= c2*nabla.T @ p:
        a *= 0.5
        x_new = x + a * p
        nabla_new = grad(f, x_new)
    return a

def grad(f, x):
    global f_count
    h = np.cbrt(np.finfo(float).eps)
    d = len(x)
    nabla = np.zeros(d)
    for i in range(d):
        x_for = np.copy(x)
        x_back = np.copy(x)
        x_for[i] += h
        x_back[i] -= h
        nabla[i] = (f(x_for) - f(x_back)) / (2*h)
        f_count += 2
    return nabla
```

Метод Нелдера-Мида

```
def nelder_mead(function, start_point, eps: float, plot=True):
  dimension = 2
  maxiter = 200
  points = [point for point in initial_points(dimension, start_point)]
  x_store, f_count, it = [], 0, 1
  for i in range(maxiter):
    points = sorted(points, key=lambda x: function(x))
    better_point = points[0]
    second_worst_point = points[len(points) - 2]
    worst_point = points[-1]
    function_in_better_point = function(better_point)
    f_count += 1
    if function == f3 and (4.260718944831057 - eps < abs(function_in_better_point) < 4.260718944831057 + eps):
      break
    elif abs(function_in_better_point) < eps:
      break
    function_in_second_worst_point = function(second_worst_point)
    f_count += 1
    center = center_of_gravity(points, worst_point)
    reflection = reflect_relatively_point(worst_point, center)
    function_in_reflection_point = function(reflection)
    f_count += 1
    if function_in_better_point <= function_in_reflection_point < function_in_second_worst_point:
      points = [point for point in points if point != worst_point]
      points.append(reflection)
    elif function_in_reflection_point < function_in_better_point:
      expanded = increase_distance_from_point(reflection, center)
```

```
      if function(expanded) < function_in_reflection_point:
        points = [point for point in points if point != worst_point]
        points.append(expanded)
      else:
        points = [point for point in points if point != worst_point]
        points.append(reflection)
        f_count += 1
    elif function_in_reflection_point >= function_in_second_worst_point:
      if function(reflection) < function(worst_point):
        contracted = reduce_distance_to_point(reflection, center)
        if function(contracted) < function_in_reflection_point:
          points = [point for point in points if point != worst_point]
          points.append(contracted)
        else:
          points = compress_all_points(points, better_point)
          f_count += 1
      else:
        contracted = reduce_distance_to_point(worst_point, center)
        if function(contracted) < function_in_reflection_point:
          points = [point for point in points if point != worst_point]
          points.append(contracted)
        else:
          points = compress_all_points(points, better_point)
          f_count += 1
      f_count += 2
    x_store.append(sorted(points, key=lambda x: function(x))[-1])
    it += 1
  x_store = np.array(x_store)
  if plot == True:
    make_plot(x_store, function, 'NELDER-MEAD')
  else:
    print('{:.0e}'.format(eps), it-1, f_count,
          '(' + str('{:.5e}'.format(x_store[-1][0])) + ', ' + str('{:.5e}'.format(x_store[-1][1])) + ')',
          '{:.3e}'.format(abs(function([x_store[-1][0], x_store[-1][1]]))))

def center_of_gravity(points: list, worst_point: list):
  dimension = len(worst_point)
  center = [0.] * dimension
  for i in range(dimension):
    center[i] = sum([point[i] for point in points
                     if point != worst_point]) / dimension
  return center

def reflect_relatively_point(point: list, symmetry_center: list, ratio=1.):
  dimension = len(point)
  reflection = [0.] * dimension
  for i in range(dimension):
    reflection[i] = symmetry_center[i] + ratio * (symmetry_center[i] - point[i])
  return reflection

def change_distance_relatively_point(moving_point: list, point: list, ratio: float):
  dimension = len(moving_point)
  new_point = [0.] * dimension
  for i in range(dimension):
    new_point[i] = point[i] + ratio * (moving_point[i] - point[i])
  return new_point

def increase_distance_from_point(moving_point: list, point: list, ratio=2.):
  return change_distance_relatively_point(moving_point, point, ratio)

def reduce_distance_to_point(moving_point: list, point: list, ratio=.5):
  return change_distance_relatively_point(moving_point, point, ratio)

def compress_all_points(points: list, better_point: list, ratio=.5):
  dimension = len(better_point)
  for i in range(dimension):
    [map(lambda x: better_point[i] + ratio * (x[i] - better_point[i]), point)
    for point in points]
  return points
```

```python
def initial_points(n, point):
  yield deepcopy(point)
  d_1 = (math.sqrt(n + 1) + n - 1) / (n * math.sqrt(2.))
  d_2 = 1. / (n * math.sqrt(2.) * (math.sqrt(n + 1) - 1.))
  number_of_current_point = 0
  while number_of_current_point < n:
    for i in range(n):
      point[i] = d_2
      if i == number_of_current_point:
        point[i] = d_1
    number_of_current_point += 1
    yield deepcopy(point)
```

6. Выводы

В методе Бройдена количество итераций для разных значений точности примерно одинаково на каждой из целевых функций, при этом более ощутим рост количества вычислений функции с каждой итерацией.

Стоимость одной итерации: $O(n^2)$ + вычисление функции и оценка градиента.
Преимущество метода заключается в отсутствии необходимости решения линейных систем уравнений и в особенности – необходимости вычисления вторых производных. В сравнении с ньютоновскими методами, метод Бройдена сходится довольно медленно, однако эффективность достигается за счёт низкой стоимости итерации.
Алгоритм устойчив и имеет сверхлинейную сходимость.
Отдельно следует упомянуть самокорректирующиеся свойства. Если гессиан неверно оценивает кривизну функции и эта плохая оценка замедляет алгоритм, тогда аппроксимация гессиана стремится исправить ситуацию за несколько шагов. Самокорректирующиеся свойства алгоритма работают благодаря реализации соответствующего линейного поиска, в котором соблюдаются условия Вольфе.
В методе Нелдера-Мида требуется большее количество итераций, но каждая итерация требует меньшее количество вычислений функции. Алгоритм сходится быстрее по сравнению с методом Бройдена, однако точность значительно снижается.
Сходимость обоих методов наступает тем быстрее, чем ближе начальное приближение к точке экстремума. При этом существуют области вблизи экстремума, не приводящие к сходимости. Для метода Нелдера-Мида вероятность найти такое начальное приближение, которое приведёт к сходимости, выше, чем для метода Бройдена.
Оба алгоритма хорошо сходятся на функции Розенброка.