Homework 2

Daniil Sherki

MSc-1, Petroleum Engineering

Question 1 (2 point)

Find minimum (i.e. both the point x^* and the function value $f^* = f(x^*)$) of function with respect to parameters a, b, c:

$$f(x) = ax^2 + bx + c \tag{1}$$

Solution

Let's find the derivative of $f(x)$.

$$f' = 2ax + b \tag{2}$$

Then we can equate derivative to zero

$$f' = 0 : 2ax + b = 0 \Rightarrow x^* = \frac{-b}{2a} \tag{3}$$

It's extremum, but we find exactly minimum. Thus, we need to make constraint on a parameter, because this parameter change the direction of the branches of the function, that is, it changes the direction of increasing/decreasing.

How we now it from simple school math, $a > 0$ it's condition to make x^* a minimum point.

Let's do more complicated math: if $f'(x) < 0$ - function is decreasing, if $f'(x) > 0$ - function is increasing. And function change direction of increasing/decreasing in extremum point. That means that we have condition to statement: $x*$ is a minimum point.

It is $f'(x^* + \mathrm{d}x) > 0$ and $f'(x^* - \mathrm{d}x) > 0$, but we know that x^* is a extremum, and we need to check only one condtion. Let's check the first one.

$$f'(x^* + \mathrm{d}x) > 0 \Rightarrow 2a(x^* + \mathrm{d}x) + b > 0 \Rightarrow 2a(\frac{-b}{2a} + \mathrm{d}x) + b > 0 \Rightarrow -b + 2a\mathrm{d}x + b > 0 :$$

$$a > 0 \tag{5}$$

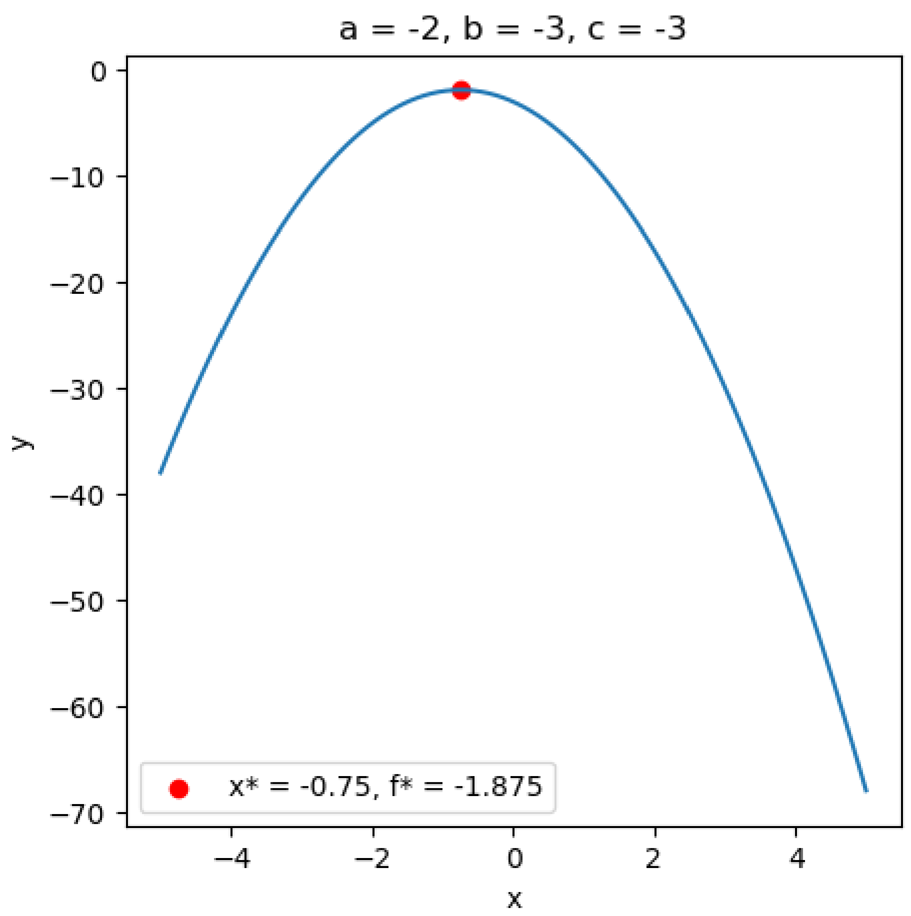
$$f^*(x^*) = ax^* + bx^* + c = a\frac{b^2}{4a^2} + b \cdot (\frac{-b}{2a}) + c = \frac{b^2}{4a} - \frac{b^2}{2a} + c \tag{6}$$

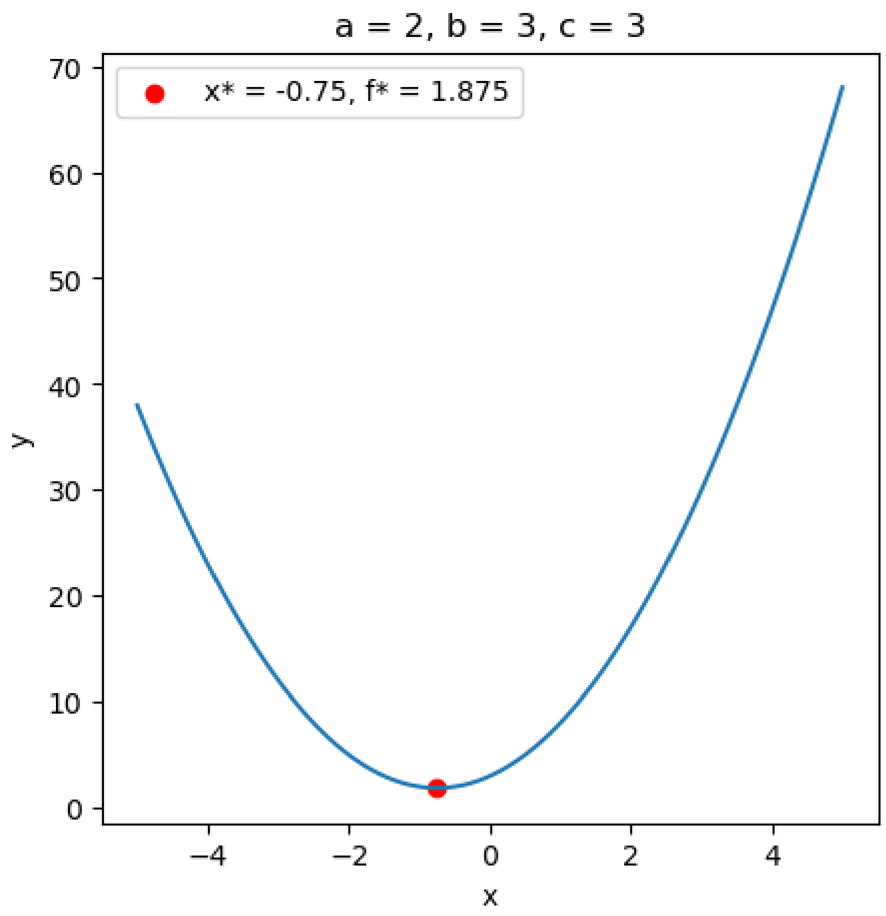
$$f^*(x^*) = -\frac{b^2}{4a} + c \tag{7}$$

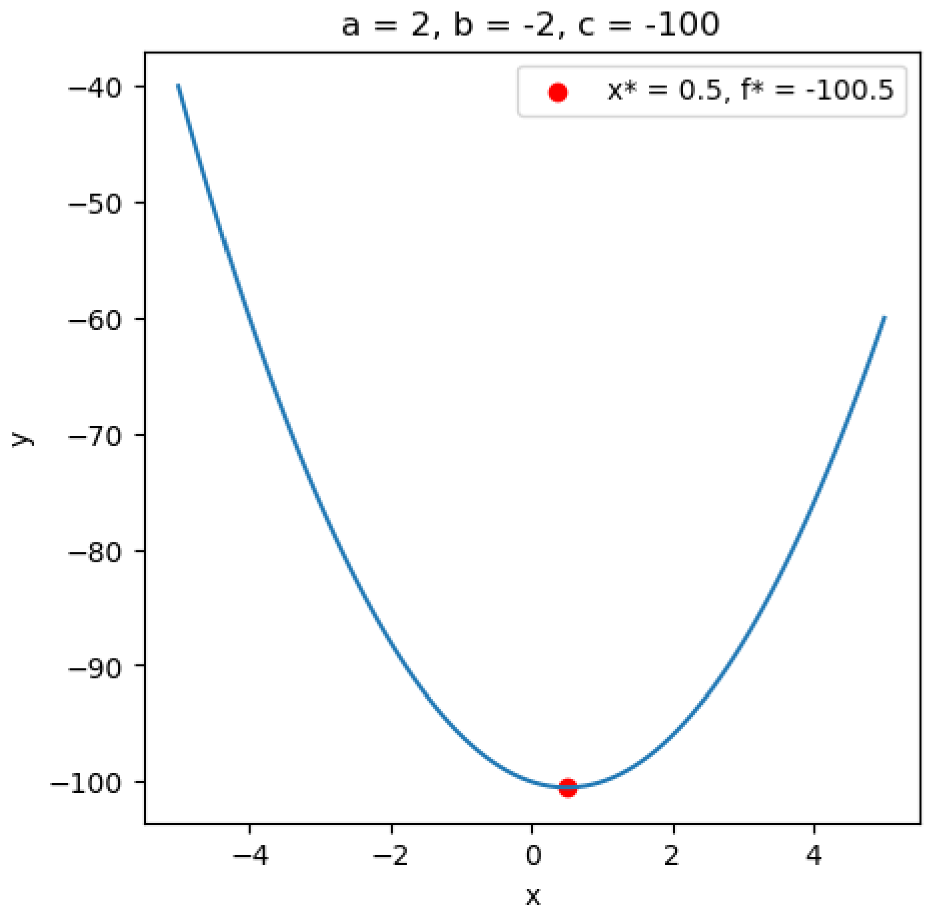
Answer: $x^* = \frac{-b}{2a}$, $f^* = -\frac{b^2}{4a} + c$ where $a > 0$

Ввод [1]:

```python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5, 5, 100)

a = [-2, 2, 2]
b = [-3, 3, -2]
c = [-3, 3, -100]

def func(x, a, b, c):
    return a * np.power(x, 2) + b * x + c

for i in range(len(a)):
    plt.figure(figsize = (5,5))
    y = func(x, a[i], b[i], c[i])
    plt.plot(x, y)
    x_star = -b[i]/(2*a[i])
    f_star = (-(b[i])**2)/(4*a[i])+c[i]
    plt.scatter(x_star, f_star, label=f'x* = {x_star}, f* = {f_star}', c=
    plt.legend()
    plt.title(f'a = {a[i]}, b = {b[i]}, c = {c[i]}')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()
```







Question 2 (1 point)

What are dimensions of gradient of a function $h(x) = f(Ax)$, constructed of function $f : \mathbb{R}^m \to \mathbb{R}$ and matrix $A \in \mathbb{R}^{m \times k}$?

Solution

There is input vector $\mathbf{x}$ with dimension $\mathbb{R}^m$ and we can represent operator $\mathbf{A}$ as $A = [a_1, \ldots, a_m]^{\mathrm{T}}$

Input vector equals: $\mathbf{x} = [x_1, \ldots, x_k]^{\mathrm{T}} \in \mathbb{R}^k$

That means that gradietns of a function wil be equal to number of variables in input vector. The gradient by definition is vector of partial derivatives of all variables.

The gradient of function $h(x)$ will be equal:

$$\nabla_x h(\mathbf{x}) = [\frac{\partial h}{\partial x_1}, \ldots, \frac{\partial h}{\partial x_k}]^{\mathrm{T}} \in \mathbb{R}^k \tag{8}$$

Ввод [2]:

```python
import numpy as np
import jax.numpy as jnp
from jax import grad

m, k = 3, 5

A = np.random.randn(m, k)
x = np.random.randn(k)
x = np.reshape(x, [k, 1])


def func(x):
    return jnp.linalg.norm(A @ x, 2)


df = grad(func)

grad_vector = df(x)

print('h(x) = f(Ax+b) = ||Ax+b||_2')
print(f'A = \n{np.round(A,2)},\n x = \n{np.round(x,2)}')
print(f'h(x) = {round(func(x),2)}')
print(f'grad h(x) = \n{np.round(grad_vector,2)}')
print(f'Gradient vector dimensions is {grad_vector.shape}')
```

```
h(x) = f(Ax+b) = ||Ax+b||_2
A = 
[[-1.01  0.69 -0.31  0.48  0.09]
 [-1.31 -1.68  0.42  0.69  0.74]
 [-0.58  1.28 -0.16 -2.21 -0.39]],
 x = 
[[ 0.18]
 [-0.86]
 [ 0.65]
 [ 0.37]
 [ 0.18]]
h(x) = 2.9800000190734863
grad h(x) = 
[[-0.13      ]
 [-2.1699998 ]
 [ 0.45999998]
 [ 1.93      ]
 [ 0.72999996]]
Gradient vector dimensions is (5, 1)
```

Question 3 (3 points)

Derive Hessian matrix $\nabla_x^2 f(x)$ for the function $f(x) = g(Ax + b)$, assuming differentiable $g : \mathbb{R}^m \to \mathbb{R}$, with dimensions $A \in \mathbb{R}^{m \times n}; b \in \mathbb{R}^m; x \in \mathbb{R}^n$.

Solution

The Hessian matrix or Hessian by definition is a square matrix of second-order partial derivatives of a scalar-valued function, or scalar field (for function of many variables).

In this case there is the same vector as in the previous task. Input vector equals:
$\mathbf{x} = [x_1, \ldots, x_k]^{\mathrm{T}} \in \mathbb{R}^n$

$$\mathbf{H}_f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n\,\partial x_1} & \dfrac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{9}$$

$$(\mathbf{H}_f)_{i,j} = \frac{\partial^2 f}{\partial x_i\,\partial x_j}. \tag{10}$$

where $i = 1, \ldots, n$, $j = 1, \ldots, n$

This is general scheme of Hessin matrix for each function.

According to chain rule and [the lecture, slide 7 (https://see.stanford.edu/materials/lsocoee364a/review7_single.pdf)](https://see.stanford.edu/materials/lsocoee364a/review7_single.pdf), we know that

$$\nabla^2 f(x) = g'(h(x))\nabla^2 h(x) + g''(h(x))\nabla f(x)\nabla f(x)^{\mathrm{T}} \tag{11}$$

In our case, when $f(x) = g(Ax + b)$

The second derivative of f is

$$\nabla^2 f(x) = A^{\mathrm{T}}\nabla^2 g(Ax + b)A \tag{12}$$

Ввод [3]:

```python
import numpy as np
import jax.numpy as jnp
from jax import hessian

m, n = 3, 4

A = np.random.randn(m, n)
b = np.random.randn(m)
x = np.random.randn(n)
x = np.reshape(x, [n, 1])
b = np.reshape(b, [m, 1])


def func(x):
    return jnp.linalg.norm(A @ x + b, 2)


H = hessian(func)
hess_m = H(x)
hess_m = jnp.reshape(hess_m, [n, n])

print('f(x) = f(Ax+b) = ||Ax+b||_2')
print(f'A = \n{np.round(A,2)},\n b = \n{np.round(b,2)},\n x = \n{np.round
print(f'f(x) = {round(func(x),2)}')
print(f'Hessian f(x) = \n{np.round(hess_m,2)}')
print(f'Hesian matrix is {hess_m.shape}')
```

```
f(x) = f(Ax+b) = ||Ax+b||_2
A =
[[ 0.03  1.38 -0.78 -0.23]
 [-2.16  1.23 -0.05  0.28]
 [ 0.51 -0.91 -1.18 -2.18]],
 b =
[[ 1.27]
 [ 0.22]
 [-0.51]],
 x =
[[ 2.37]
 [-0.37]
 [ 0.03]
 [-1.44]]
f(x) = 7.179999828338623
Hessian f(x) =
[[ 0.11        -0.06        0.14        0.19      ]
 [-0.06         0.35999998 -0.14999999  0.01      ]
 [ 0.14        -0.14999999  0.19999999  0.22      ]
 [ 0.19         0.01        0.22        0.35999998]]
Hesian matrix is (4, 4)
```

Question 4 (3 points)

Solve an optimal step size problem for the quadratic function, with symmetric positive definite matrix $A \succ 0 \in \mathbb{R}^{n \times n}$, and $x, b, d \in \mathbb{R}^n$. Your goal is to find optimal γ^* for given A, b, d, x. The resulting expression must be written in terms of inner products $(\ldots, \ldots)$

$$f(\gamma) = (A(x + \gamma d), x + \gamma d) + (b, x + \gamma d) \to \min_{\gamma \in \mathbb{R}} \tag{13}$$

Solution

$$\nabla_\gamma f = \nabla_\gamma (A(x + \gamma d), x + \gamma d) + \nabla_\gamma (b, x + \gamma d) \tag{14}$$

$$\nabla_\gamma f = (\nabla_\gamma(A(x+\gamma d)), x+\gamma d) + (A(x+\gamma d), \nabla_\gamma(x+\gamma d)) + (\nabla_\gamma b, x+\gamma d) + (b, \nabla_\gamma(x+\gamma$$

$$\nabla_\gamma f = (Ad, x+\gamma d) + (A(x+\gamma d), d) + (0, x+\gamma d) + (b, d) \tag{16}$$

$$\nabla_\gamma f = (Ad, x) + (Ad, \gamma d) + (Ax, d) + (A\gamma d, d) + (0, x+\gamma d) + (b, d) \tag{17}$$

$$\nabla_\gamma f = (Ad, x) + \gamma(Ad, d) + (Ax, d) + \gamma(Ad, d) + (b, d) \tag{18}$$

$$\nabla_\gamma f = \gamma((Ad, d) + (Ad, d)) + ((Ad, x) + (Ax, d) + (b, d)) \tag{19}$$

We know that optimum point will be achieve when $\nabla_\gamma f = 0$

$$\gamma((Ad, d) + (Ad, d)) + ((Ad, x) + (Ax, d) + (b, d)) = 0 \tag{20}$$

$$\gamma = -\frac{(Ad, x) + (Ax, d) + (b, d)}{(Ad, d) + (Ad, d)} \tag{21}$$

$$\gamma = -\frac{((A + A^\top)d, x) + (b, d)}{2(Ad, d))} \tag{22}$$

A - positive-definite (e.g. symmetric) $\Rightarrow A = A^\top$

$$\gamma = -\frac{2(Ad, x) + (b, d)}{2(Ad, d)} \tag{23}$$

$$\nabla_\gamma f = 2(Ad, d)\gamma + ((Ad, x) + (Ax, d) + (b, d)) \tag{24}$$

$$\nabla^2_\gamma f = \nabla^2_\gamma(2(Ad, d)\gamma) + \nabla^2_\gamma(((Ad, x) + (Ax, d) + (b, d))) \tag{25}$$

$$\nabla^2_\gamma f = 2(Ad, d) = 2d^\top Ad \tag{26}$$

$$\nabla^2_\gamma f = 2A \tag{27}$$

$$\nabla^2_\gamma f \succ 0 \text{ so } A \succ 0 \text{ (PSD)} \tag{28}$$

Question 5 (3 points)

Derive subgradient (subdifferential) for the function $f(x) = [x^2 - 1]_+, x \in \mathbb{R}$. (Do not write the subgradient method.)

Solution

$$\partial f(x) = \begin{cases} -2x, \ x \leq -1 \\ [-2x; 2x], \ \ -1 < x < 1 \\ 2x, \ x \geq 1 \end{cases}$$

Question 6 (5 points)

Find the minimizer of

$$f(x, y) = x^2 + xy + 10y^2 - 22y - 5x \tag{29}$$

numerically by steepest descent.

1. For each iteration, record the values of x,y, and f and include in a table.
2. Plot the values on a contour plot.

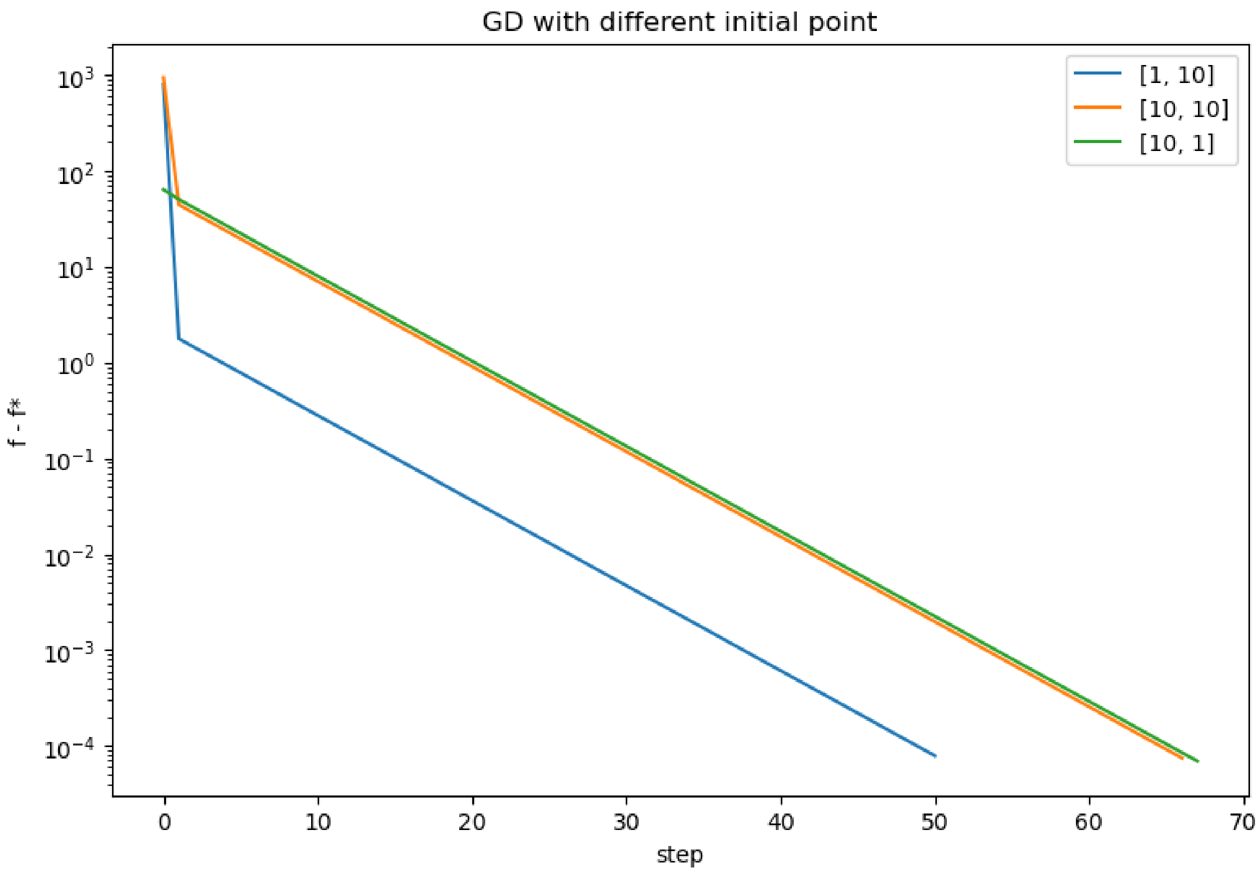
3. Explore different starting values, such as (1, 10), (10, 10), (10, 1). Does the number of steps depend significantly on the starting guess?

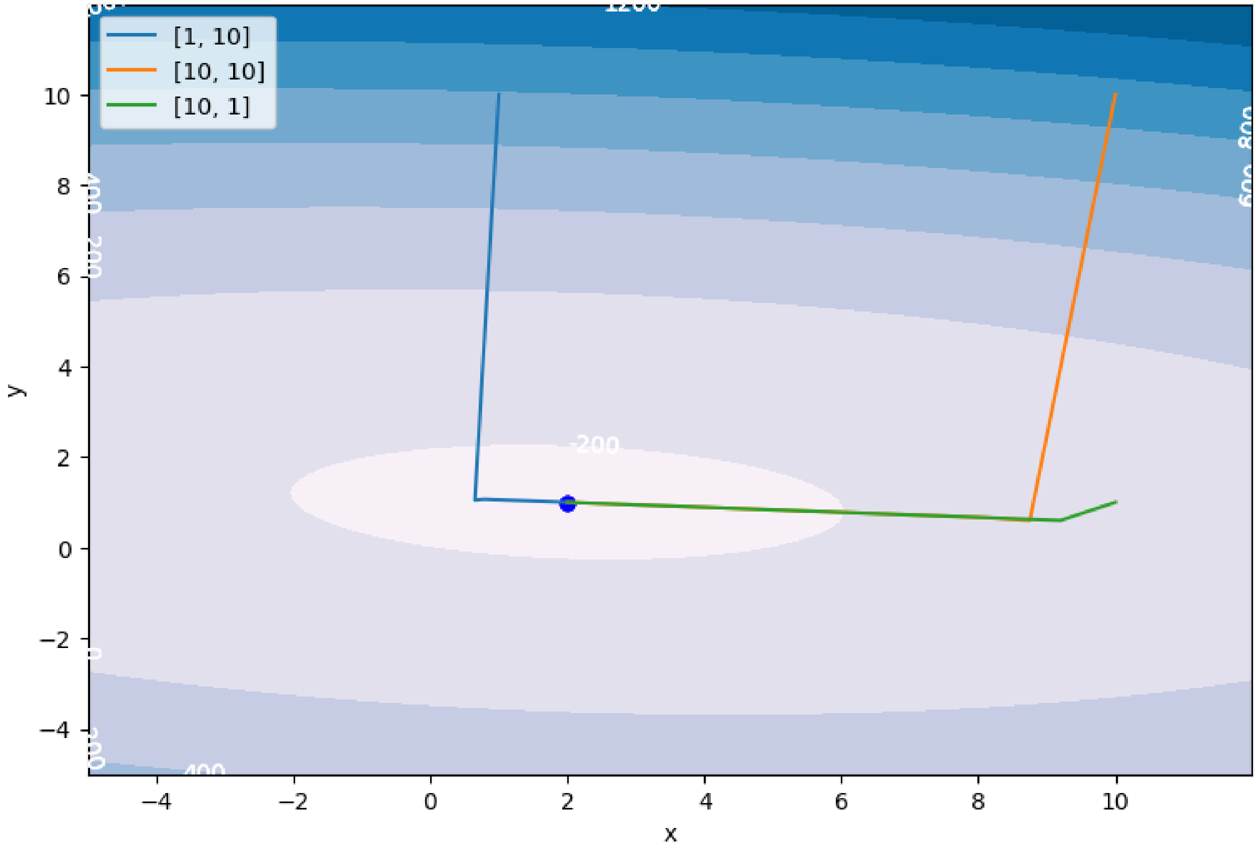
Ввод [4]:

```
import jax
import pandas as pd
import matplotlib.pyplot as plt
from scipy.optimize import fminbound


def task5_func(x):
    return np.power(x[0], 2) + x[0]*x[1] + 10 * np.power(x[1], 2) - 22*x[


def task5_grad(x):
    dfdx = 2 * x[0] + x[1] - 5
    dfdy = x[0] + 2*10 * x[1] - 22
    return np.array([dfdx, dfdy])


def gradient_descent(x0, gamma, tol, func=task5_func, grad_func=task5_gra
    x = x0
    x_prev = np.zeros_like(x)
    k = 0

    x_arr = [x[0]]
    y_arr = [x[1]]
    f_arr = [func(x)]
    k_arr = [k]
    gamma_arr = [gamma]

    while np.linalg.norm(x - x_prev, 2) > tol:
        x_prev = x
        if adapt_gamma:
            def gamma_adapt(gamma):
                dx = 2 * x[0] + x[1] - 5
                dy = x[0] + 2*10 * x[1] - 22
                xx = x[0] - gamma*(2 * x[0] + x[1] - 5)
                yy = x[1] - gamma*(x[0] + 2*10 * x[1] - 22)
                z = int(xx**2 + xx*yy + 10 * yy**2 - 22*yy - 5*xx)
                return z
            gamma = fminbound(gamma_adapt, 0, gamma_max,
                              xtol=1e-02, maxfun=100)
            gamma_arr.append(gamma)
        k += 1
        x = x - gamma*grad_func(x)

        x_arr.append(x[0])
        y_arr.append(x[1])
        f_arr.append(func(x))
        k_arr.append(k)

    if adapt_gamma:
        df = pd.DataFrame({'step': k_arr, 'x': x_arr,
                           'y': y_arr, 'f': f_arr, 'gamma': gamma_arr})
    else:
        df = pd.DataFrame({'step': k_arr, 'x': x_arr, 'y': y_arr, 'f': f_
    return x, func(x), k, df
```

Ввод [5]:

```
init_points = [[1, 10], [10, 10], [10, 1]]

xmin, xmax, step = -5, 12, 0.01
x = np.arange(xmin, xmax, step)
y = np.arange(xmin, xmax, step)
xgrid, ygrid = np.meshgrid(x, y)
zgrid = task5_func([xgrid, ygrid])

fig = plt.figure(figsize = (9, 6))
ax1 = fig.add_subplot()
fig = plt.figure(figsize = (9, 6))
ax2 = fig.add_subplot()
count_chart = ax2.contourf(xgrid, ygrid, zgrid, cmap='PuBu')
for init in init_points:
    _, y_star, _, df = gradient_descent(
        np.array(init), 0.05, 1e-3, adapt_gamma=False)
    df['f_fst'] = df['f']+16
    ax1.plot(df.step, df.f_fst, label=f'{init}')
    ax2.plot(df.x, df.y, label=f'{init}')
    plt.clabel(count_chart, inline=0, fontsize=10, colors='white', fmt='%1
    ax2.scatter(2, 1, c='b')
    ax1.set_yscale('log')
ax1.set_xlabel('step')
ax1.set_ylabel('f - f*')
ax1.set_title(f'GD with different initial point')
ax1.legend()
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.legend()
plt.show()
```

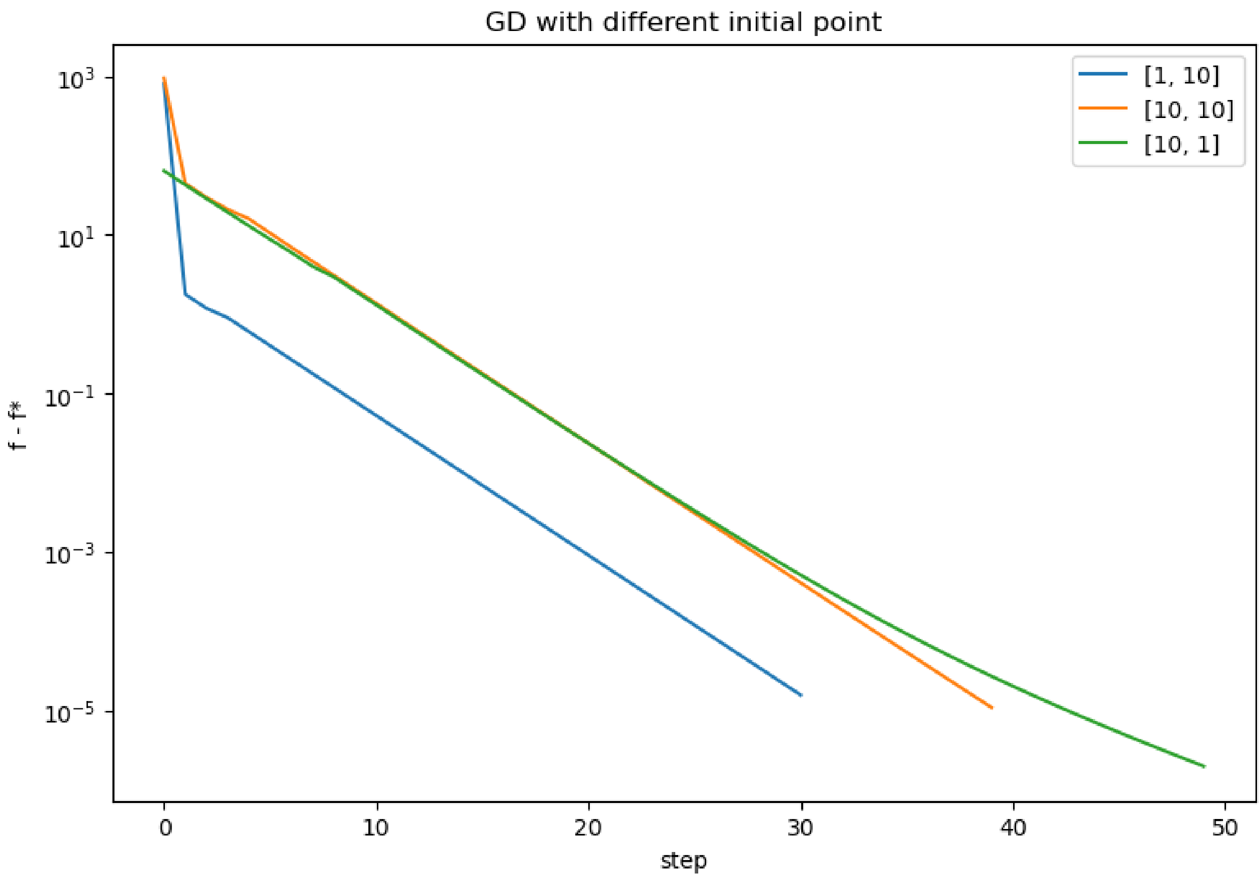


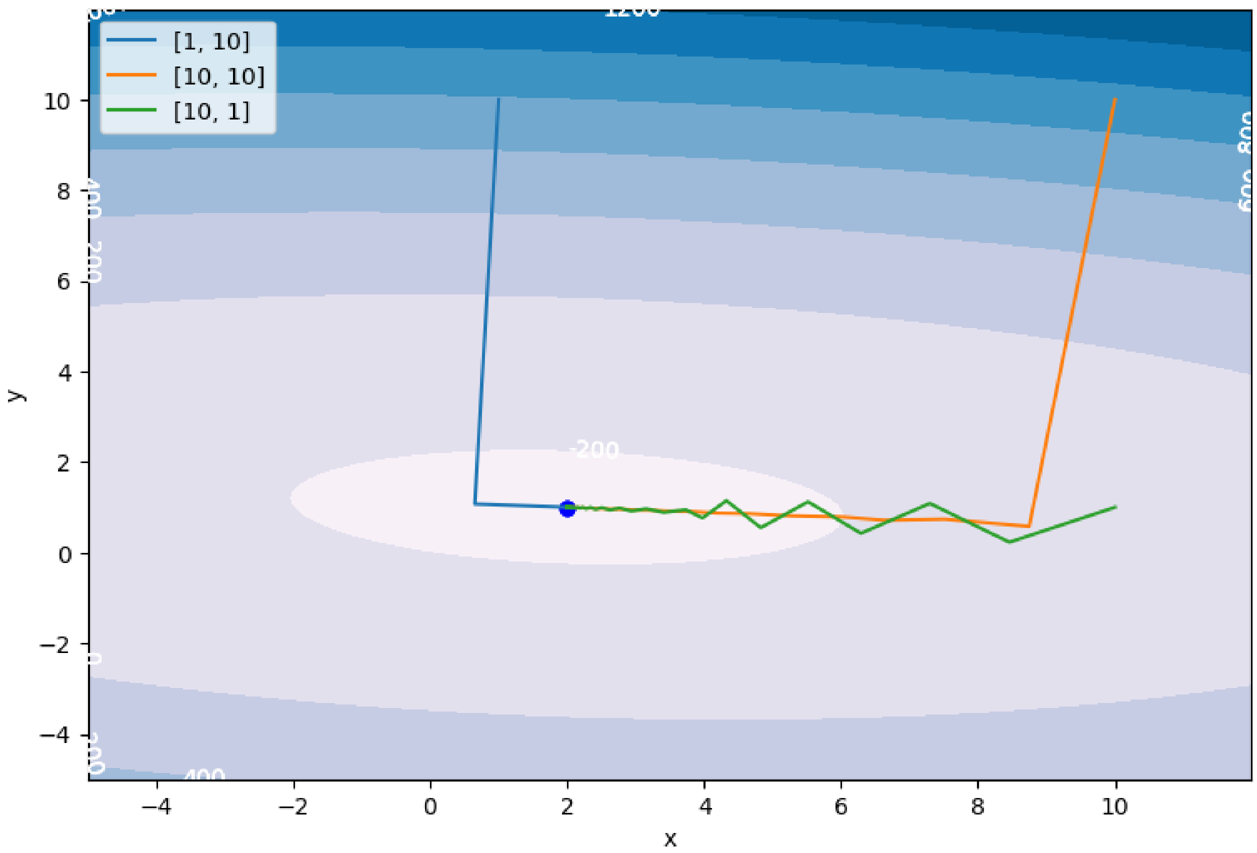


[1, 10]
[10, 10]
[10, 1]
1200
800
600
400
200
-200
0
y
x

Ввод [6]:

```
init_points = [[1, 10], [10, 10], [10, 1]]

xmin, xmax, step = -5, 12, 0.01
x = np.arange(xmin, xmax, step)
y = np.arange(xmin, xmax, step)
xgrid, ygrid = np.meshgrid(x, y)
zgrid = task5_func([xgrid, ygrid])

fig = plt.figure(figsize = (9, 6))
ax1 = fig.add_subplot()
fig = plt.figure(figsize = (9, 6))
ax2 = fig.add_subplot()
count_chart = ax2.contourf(xgrid, ygrid, zgrid, cmap='PuBu')
for init in init_points:
    _, y_star, _, df = gradient_descent(
        np.array(init), 0.05, 1e-3, adapt_gamma=True, gamma_max=0.1)
    df['f_fst'] = df['f']+16
    ax1.plot(df.step, df.f_fst, label=f'{init}')
    ax2.plot(df.x, df.y, label=f'{init}')
    plt.clabel(count_chart, inline=0, fontsize=10, colors='white', fmt='%
    ax2.scatter(2, 1, c='b')
    ax1.set_yscale('log')
ax1.set_xlabel('step')
ax1.set_ylabel('f - f*')
ax1.set_title(f'GD with different initial point')
ax1.legend()
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.legend()
plt.show()
```





Ввод [7]:

```
import plotly.express as px

px.line(data_frame=df, x = 'step', y =  'gamma', title = 'Gamma changing
```

Question 7 (9 points)

Let the cost function of the unconstrained optimization problem of interest be:

$$f(x) = \frac{1}{4}(x_1 - 1)^2 + \sum_{i=2}^{n}(2x_{i-1}^2 - x_i - 1)^2 \tag{30}$$

Consider two different scenarios, first $n = 3$, and then $n = 10$. Recall that the steepest descent algorithm is,

$$x^{k+1} = x^k - \alpha^k \nabla f(x^k), \alpha^k \in \mathbb{R}_{>0} \tag{31}$$

1. Using $x^0 = [-1.5, 1, \dots, 1]^\top$ write out the first iteration of the steepest descent algorithm and obtain the optimum value for α^0. What is the value of x^1 if you implement α^0? Verify that $f(x^1) < f(x^0)$.
2. Write a code to find the minimizer of f(x) using steepest descent algorithm with starting point of $x^0 = [-1.5, 1, \dots, 1]^\top$ and using

- $\alpha^k = \text{argmin } f(x^k - \alpha \nabla f(x^k))$
- constant $\alpha = 0.1$
- constant $\alpha = 0.5$
- constant $\alpha = 1.0$ (Use $\|x^{k+1} - x^k\| \leq 10^{-6}$ as the stopping condition for your algorithm.)

3. How many steps does it take for the algorithm to converge for each choice of the step size above?
4. Use fminbnd from Matlab or an equivalent function from the programming language of your choice to solve the problem. How many steps does it take for this algorithm to converge?

Ввод [8]:

```
import jax.numpy as jnp

def task7_func(x):
    sum_x = 0
    for i in range(1,len(x)):
        sum_x += jnp.power(2*jnp.power(x[i-1],2)-x[i]-1,2)
    out = 0.25*jnp.power(x[0] - 1,2) + sum_x
    return out
```

Ввод [9]:

```
import numpy as np

def task7_func_analyt(x):
    sum_x = 0
    for i in range(1,len(x)):
        sum_x += np.power(2*np.power(x[i-1],2)-x[i]-1,2)
    out = 0.25*np.power(x[0] - 1,2) + sum_x
    return out
```

Ввод [10]:

```
x3 = np.array([-1.5,1,1])
x10 = np.array([-1.5,1,1,1,1,1,1,1,1,1])
```

Ввод [11]:

```
def task7_grad_analyt(x):
  grad_vec = np.zeros_like(x)
  grad_vec[0] = 0.5 * (x[0]-1) + 16*x[0]**3 - 8 * x[0]*x[1] - 8*x[0]
  for i in range(1, len(grad_vec)-1):
    grad_vec[i] = 2*x[i] + 2 - x[i-1]**2 * 4 + x[i]**3 * 16 - 8 * x[i]*x[
  j = len(grad_vec)-1
  grad_vec[j] = 2*x[j] + 2 - 4 * x[j-1]**2
  return grad_vec
task7_grad_analyt(x3), task7_grad_analyt(x10)
```

Out[11]:

```
(array([-31.25,  -5.  ,   0.  ]),
 array([-31.25,  -5.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
          0.  ,   0.  ]))
```

Ввод [12]:

```
task7_grad = jax.grad(task7_func)
task7_grad(x3), task7_grad(x10)
```

Out[12]:

```
(DeviceArray([-31.25,  -5.  ,   0.  ], dtype=float32),
 DeviceArray([-31.25,  -5.  ,   0.  ,   0.  ,   0.  ,   0.  ,   0.  ,
                0.  ,   0.  ,   0.  ], dtype=float32))
```

Ввод [13]:

```
def jax_gradient_descent(x0, gamma, tol, func=task7_func_analyt, grad_func
                         adapt_gamma=False, gamma_max=1, adapt_auto=True,
    x = x0
    x_prev = np.zeros_like(x)
    k = 0
    k1 = 0
    brut_force = False

    x_arr = [x]
    f_arr = [func(x)]
    k_arr = [k]
    k1_arr  = [k1]
    gamma_arr = [gamma]

    while np.linalg.norm(x - x_prev, 2) > tol:
        x_prev = x
        if adapt_gamma:
            if adapt_auto:
                def gamma_adapt(gamma):
                    z = func(x - gamma*grad_func(x))
                    return z
                gamma,_,_,k1 = fminbound(gamma_adapt, 0, gamma_max,
                                   xtol=1e-05, maxfun=100, full_output=True
                k1_arr.append(k1)
                gamma_arr.append(gamma)
                print(x)
            else:
                k1 = 0
                gamma = gamma_max
                grad_val = grad_func(x)
                while func(x - gamma*grad_val)>(func(x) - c1 * gamma * np
                    gamma -= gamma_step
                    k1 += 1
                k1_arr.append(k1)
                gamma_arr.append(gamma)
        k += 1
        x = x - gamma*grad_func(x)

        x_arr.append(x)
        f_arr.append(func(x))
        k_arr.append(k)

    if adapt_gamma:
        df = pd.DataFrame({'step': k_arr, 'f': f_arr, 'gamma': gamma_arr,
    else:
        df = pd.DataFrame({'step': k_arr, 'f': f_arr})
    return x, func(x), k, df, x_arr
```

##1 subtask

Ввод [14]:

```
x3 = np.array([-1.5, 1, 1])
y3_0 = task7_func_analyt(x3)
x10 = np.array([-1.5, 1, 1, 1, 1, 1, 1, 1, 1, 1])
y10_0 = task7_func_analyt(x10)

gamma = 1
grad_val = task7_grad_analyt(x3)
while task7_func_analyt(x3 - gamma*grad_val) > (task7_func_analyt(x3) - 0
    gamma -= 0.0005
print(gamma)

x3 = x3 - gamma*task7_grad_analyt(x3)

print('3-dim case\n')
print(f'x1 = {x3}')
if task7_func(x3) < y3_0:
    print(f'f(x1)<f(x0), {task7_func_analyt(x3):.2f}<{y3_0:.2f}\n')

gamma = 1
grad_val = task7_grad_analyt(x10)
while task7_func_analyt(x10 - gamma*grad_val) > (task7_func_analyt(x10) -
    gamma -= 0.0005
print(gamma)

print('10-dim case\n')
x10 = x10 - gamma*task7_grad_analyt(x10)
print(f'x1 = {x10}')
if task7_func(x10) < y10_0:
    print(f'f(x1)<f(x0), {task7_func_analyt(x10):.2f}<{y10_0:.2f}')
```

```
0.0915000000000547
3-dim case

x1 = [1.359375 1.4575   1.       ]
f(x1)<f(x0), 6.62<7.81

0.0915000000000547
10-dim case

x1 = [1.359375 1.4575   1.        1.        1.        1.        1.        1.
 1.        1.       ]
f(x1)<f(x0), 6.62<7.81
```

Solution for 3 points

Case 1: $\alpha^k = \text{argmin } f(x^k - \alpha \nabla f(x^k))$ (manually)

Ввод [15]:

```
x = np.array([-1.5,1,1])
_, _, _, df_3c1,x_arr = jax_gradient_descent(x, 0.5, 1e-6,
                                              adapt_gamma=True,
                                              adapt_auto=False,
                                              c1=0.01,
                                              gamma_max = 0.5,
                                              gamma_step = 0.005)
```

Ввод [16]:

```
df_3c1
```

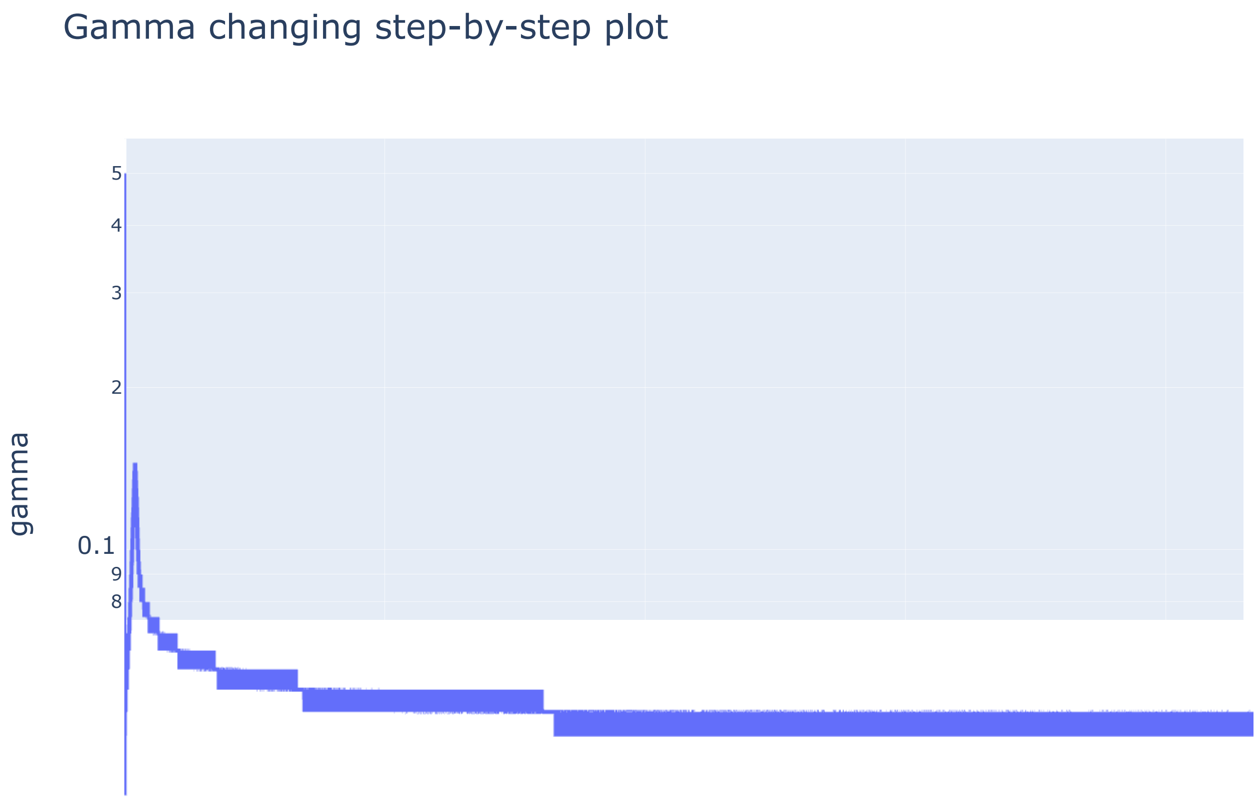
Out[16]:

	step	f	gamma	k1
0	0	7.812500e+00	0.500	0
1	1	5.877086e+00	0.090	82
2	2	4.266347e+00	0.125	75
3	3	3.763706e+00	0.035	93
4	4	3.053563e+00	0.075	85
...	...	...	...	...
63465	63465	8.256501e-09	0.045	91
63466	63466	8.256039e-09	0.050	90
63467	63467	8.253617e-09	0.045	91
63468	63468	8.253144e-09	0.050	90
63469	63469	8.250736e-09	0.045	91

63470 rows × 4 columns

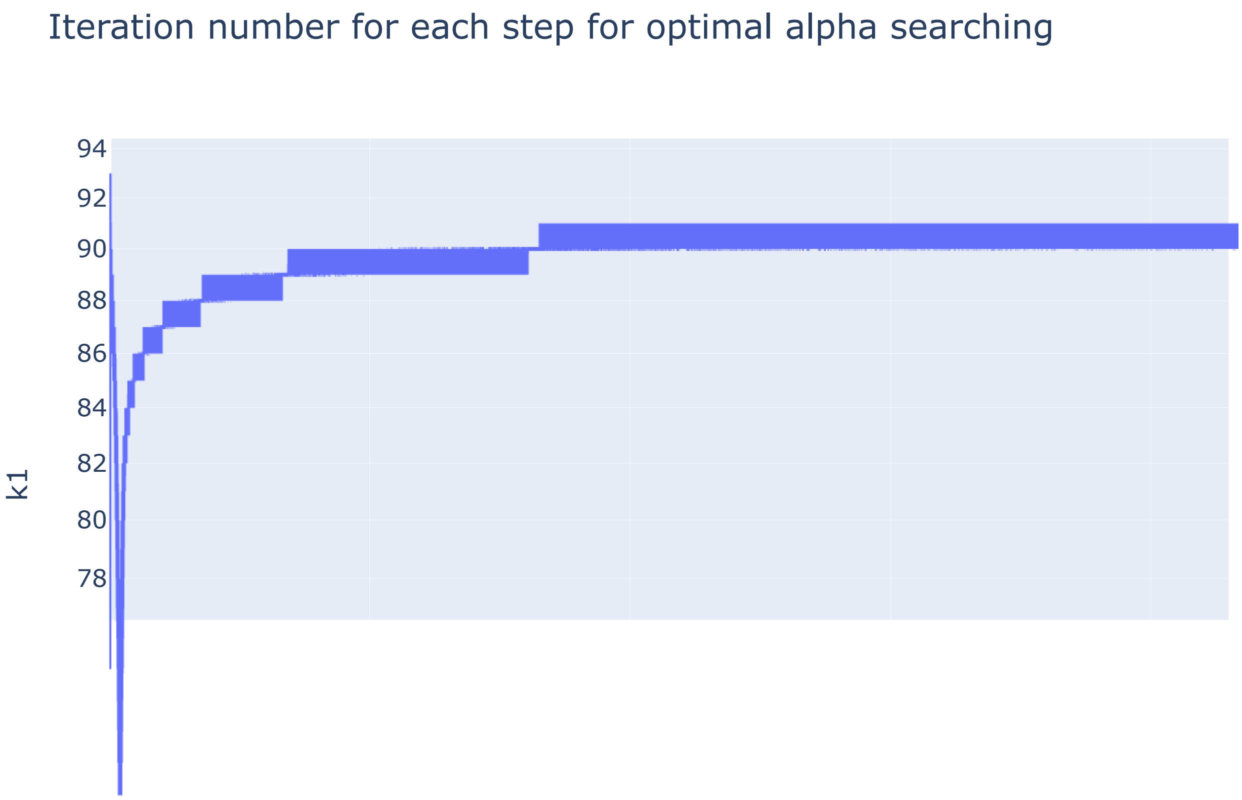
Ввод [17]:

```
import plotly.express as px

px.line(data_frame=df_3c1, x = 'step', y =  'gamma',
        title = 'Gamma changing step-by-step plot',
        log_y = True)
```



Ввод [18]:

```
px.line(data_frame=df_3c1, x = 'step', y =  'k1',
        title = 'Iteration number for each step for optimal alpha searchi
        log_y = True)
```



Ввод [19]:

```
px.line(data_frame=df_3c1, x = 'step', y =  'f',
        title = 'Function value for each step',
        log_y = True)
```

Ввод [20]:

```
x_arr[-1]
```

Out[20]:

```
array([0.99981956, 0.9992686 , 0.99707135])
```

Case 2: $\alpha^k = \text{argmin}\ f(x^k - \alpha\nabla f(x^k))$ (fminbound)

Ввод [21]:

```
x = np.array([-1.5,1,1])

_, _, _, df_3c2,x_arr = jax_gradient_descent(x, 0.5, 1e-6,
                                             adapt_gamma=True,
                                             adapt_auto=True)
```

```
[-1.5  1.   1. ]
[0.98695921 1.39791347 1.        ]
[1.04570933 1.0306176  1.06302993]
[1.0046167  1.02466794 1.06681531]
[1.00580381 1.01753755 1.06847939]
[1.00419723 1.01730478 1.0686289 ]
[1.004242   1.01703451 1.06868931]
[1.00418028 1.01702472 1.06869123]
[1.00418215 1.01701281 1.06869053]
[1.00417462 1.01701222 1.06868028]
[1.00417993 1.01700771 1.06867663]
[1.00417595 1.01701058 1.06866727]
[1.00417932 1.01700472 1.06866404]
[1.00417507 1.01700741 1.06865475]
[1.00417855 1.01700164 1.06865149]
[1.00417433 1.01700433 1.0686422 ]
[1.0041778  1.01699855 1.06863894]
[1.00417357 1.01700125 1.06862965]
[1.00417704 1.01699547 1.06862639]
```

Ввод [22]:

```
df_3c2
```

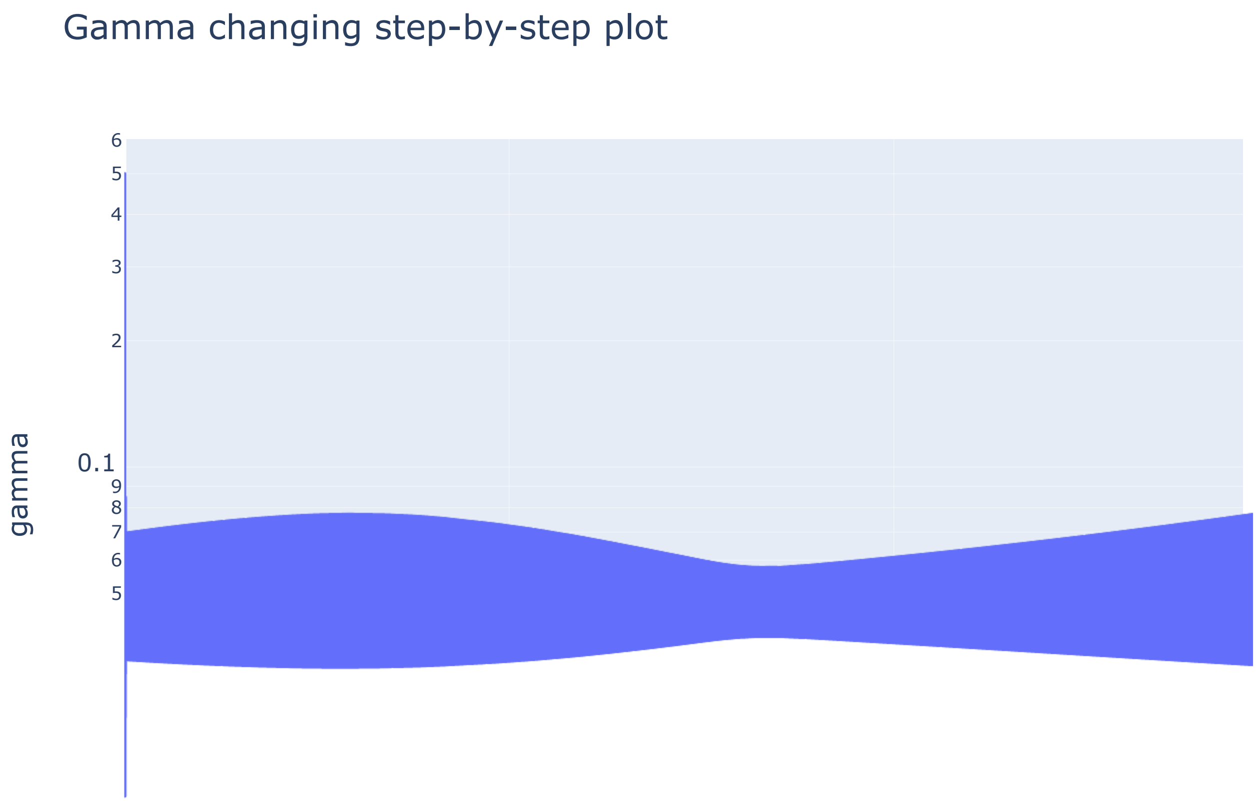
Out[22]:

	step	f	gamma	k1
0	0	7.812500e+00	0.500000	0
1	1	3.844007e+00	0.079583	16
2	2	2.874237e-02	0.016514	21
3	3	1.137110e-03	0.030868	12
4	4	4.665201e-05	0.025157	11
...	...	...	...	...
21460	21460	6.197278e-08	0.029148	7
21461	21461	6.195562e-08	0.125281	6
21462	21462	6.193846e-08	0.029147	7
21463	21463	6.192130e-08	0.125300	6
21464	21464	6.190415e-08	0.029146	7

21465 rows × 4 columns

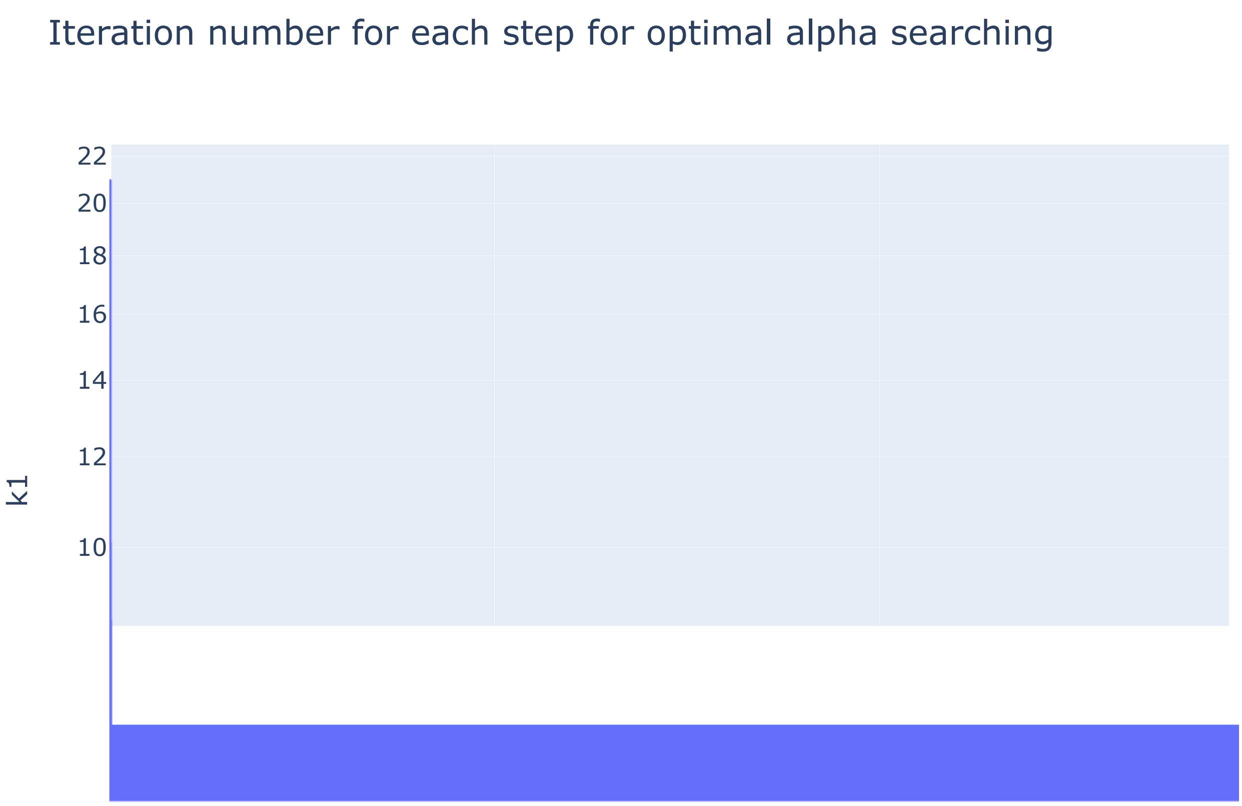
Ввод [23]:

```
1  px.line(data_frame=df_3c2, x = 'step', y =  'gamma', title = 'Gamma chang
```



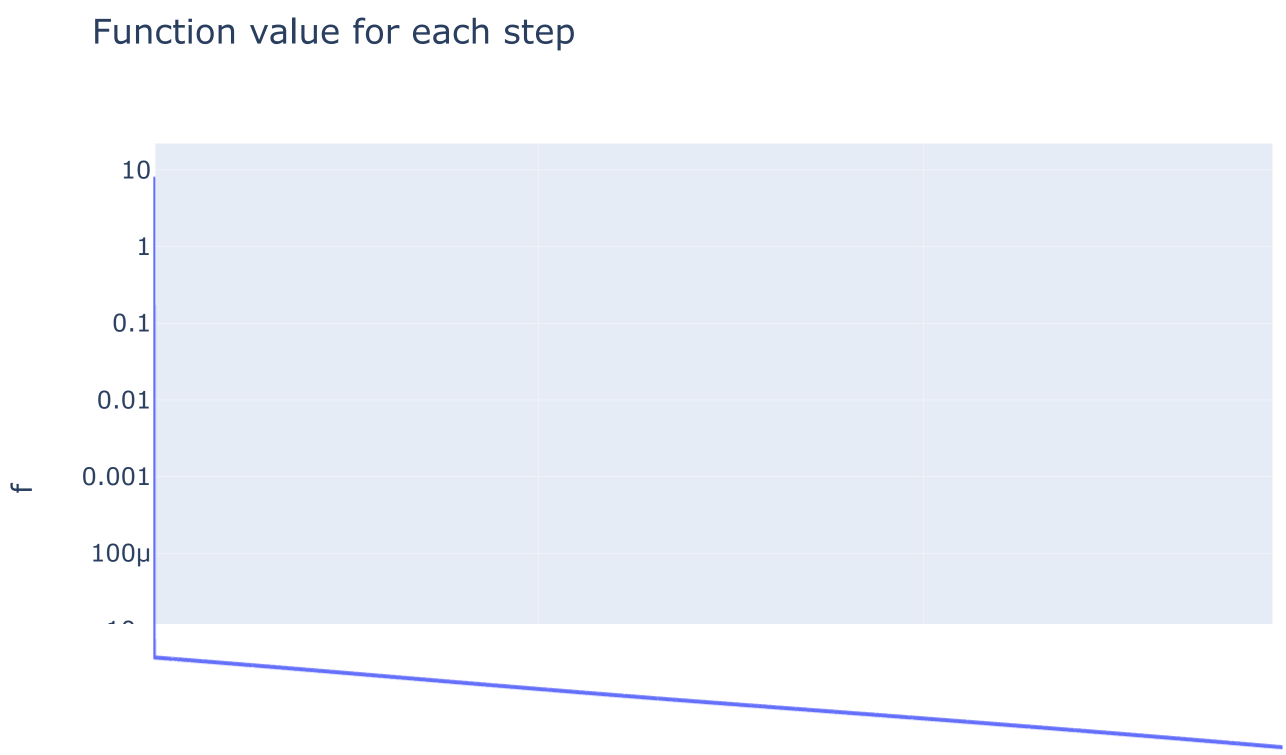
Ввод [24]:

```
px.line(data_frame=df_3c2, x = 'step', y =  'k1', title = 'Iteration numbe
```



Ввод [25]:

```
1  px.line(data_frame=df_3c2, x = 'step', y =  'f', title = 'Function value :
```



Ввод [26]:

```
1  x_arr[-1]
```

Out[26]: array([1.0004937 , 1.00200544, 1.00803755])

Case 3: $\alpha = 0.1$

Ввод [27]:
```
x = np.array([-1.5,1,1])
_, _, _, df_3c3,x_arr = jax_gradient_descent(x, 0.1, 1e-6, adapt_gamma=Fa
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:7: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:7: RuntimeWarning:

overflow encountered in double_scalars
```

Ввод [28]:

```
1 df_3c3
```

Out[28]:

	step	f
0	0	7.812500e+00
1	1	1.408301e+01
2	2	6.872906e+01
3	3	6.382117e+04
4	4	1.052642e+14
5	5	4.777552e+41
6	6	4.466592e+124
7	7	inf
8	8	inf
9	9	NaN

Ввод [29]:

```
1 x_arr
```

Out[29]:

```
[array([-1.5,  1. ,  1. ]),
 array([1.625, 1.5  , 1.   ]),
 array([-2.021875, -0.94375 ,  1.5      ]),
 array([11.26284487,  0.13759414,  1.35626563]),
 array([-2.26493743e+03,  5.09059439e+01,  8.92585359e-01]),
 array([1.85902985e+10, 1.84102474e+06, 1.03708012e+03]),
 array([-1.02796677e+31,  1.28255811e+20,  1.35574884e+12]),
 array([1.73802975e+93, 3.88930257e+61, 6.57982125e+39]),
 array([-8.40023825e+279,  1.11416743e+186,  6.05066981e+122]),
 array([nan, nan, inf])]
```

Case 4: $\alpha = 0.5$

Ввод [30]:

```
1  _, _, _, df_3c4,x_arr = jax_gradient_descent(x, 0.5, 1e-6, adapt_gamma=Fa
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:7: RuntimeWarning:

overflow encountered in double_scalars
```

Ввод [31]:

```
1  df_3c4
```

Out[31]:

	step	f
0	0	7.812500e+00
1	1	1.562042e+05
2	2	9.856757e+17
3	3	2.451558e+56
4	4	3.771956e+171
5	5	inf
6	6	inf
7	7	NaN

Ввод [32]:
```
x_arr
```

Out[32]:
```
[array([-1.5,  1. ,  1. ]),
 array([14.125,  3.5  ,  1.   ]),
 array([-22280.171875,     83.03125 ,     23.5      ]),
 array([8.84800910e+13, 9.88240789e+08, 1.37873770e+04]),
 array([-5.54149146e+42,  7.93636837e+27,  1.95323971e+18]),
 array([1.36135062e+129, 5.74172181e+085, 1.25971886e+056]),
 array([           -inf, 2.19223529e+258, 6.59347386e+171]),
 array([nan, nan, inf])]
```

Case 5: $\alpha = 1$

Ввод [33]:
```
_, _, _, df_3c5,x_arr = jax_gradient_descent(x, 1, 1e-6, adapt_gamma=False
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:7: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars
```

Ввод [34]:
```
df_3c5
```

Out[34]:

	step	f
0	0	7.812500e+00
1	1	3.113716e+06
2	2	1.240043e+23
3	3	7.810336e+72
4	4	1.951502e+222
5	5	inf
6	6	NaN

Ввод [35]:
```
x_arr
```

Out[35]:
```
[array([-1.5,  1. ,  1. ]),
 array([29.75,  6.   ,  1.   ]),
 array([-4.19608375e+05,  1.72250000e+02,  1.41000000e+02]),
 array([1.18209512e+18, 7.04203178e+11, 1.18537250e+05]),
 array([-2.64287887e+55,  1.94197830e+33,  1.98360847e+24]),
 array([2.95360056e+167, 2.79392350e+111, 1.50851189e+067]),
 array([            -inf,              nan, 3.12240341e+223])]
```

How many iterations fminbound do

Ввод [36]:
```
def gamma_adapt(gamma):
    z = task7_func(x - gamma*task7_grad(x))
    return z
gamma = fminbound(gamma_adapt, 0, 1,
                  xtol=1e-05, maxfun=100, disp=3)

```

```
 Func-count     x          f(x)          Procedure
    1        0.381966      46010.8        initial
    2        0.618034       397401        golden
    3        0.236068      4405.23        golden
    4        0.263758      7758.98        parabolic
    5        0.206276      2149.55        parabolic
    6        0.127486      105.993        golden
    7        0.138594      192.638        parabolic
    8        0.118485      60.3616        parabolic
    9       0.0732276      4.27469        golden
   10       0.0870785      4.80913        parabolic
   11       0.0796484      3.84407        parabolic
   12       0.0787963      3.85209        parabolic
   13       0.0796769      3.84413        parabolic
   14       0.0795784      3.84401        parabolic
   15       0.0795836      3.84401        parabolic
   16       0.0795917      3.84401        parabolic
   17       0.0796134      3.84402        golden
```

Answer: in worst case - 19

Solution for 10 points

Ввод [37]:
```
x = np.array([-1.5,1,1,1,1,1,1,1,1,1])
```

Case 1: $\alpha^k = \text{argmin } f(x^k - \alpha \nabla f(x^k))$ (manually)

Ввод [38]:
```
_, _, _, df_10c1,x_arr = jax_gradient_descent(x, 0.5, 1e-6, adapt_gamma=T
                                              c1 = 0.05, gamma_step = 0.0
```

Ввод [39]:

```
df_10c1
```

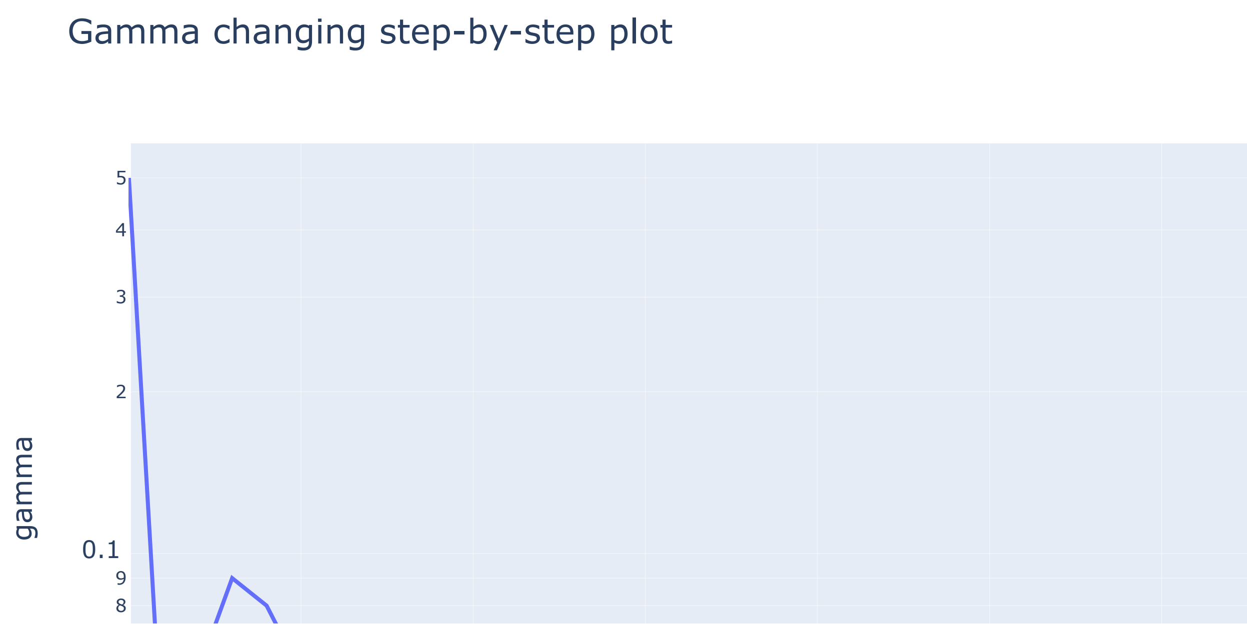
Out[39]:

	step	f	gamma	k1
0	0	7.812500	0.500	0
1	1	5.470650	0.040	192
2	2	4.696311	0.060	188
3	3	4.259271	0.090	182
4	4	3.579482	0.080	184
5	5	2.773757	0.060	188
6	6	2.458375	0.050	190
7	7	1.737589	0.040	192
8	8	1.443315	0.040	192
9	9	1.314666	0.040	192
10	10	1.249717	0.040	192
11	11	1.107408	0.035	193
12	12	1.042219	0.035	193
13	13	1.012013	0.035	193
14	14	0.997518	0.035	193
15	15	0.990364	0.035	193
16	16	0.986770	0.035	193
17	17	0.984946	0.035	193
18	18	0.984014	0.035	193
19	19	0.983537	0.035	193
20	20	0.983291	0.035	193
21	21	0.983165	0.035	193
22	22	0.983100	0.035	193
23	23	0.983067	0.035	193
24	24	0.983050	0.035	193
25	25	0.983041	0.035	193
26	26	0.983036	0.035	193
27	27	0.983034	0.035	193
28	28	0.983032	0.035	193
29	29	0.983032	0.035	193
30	30	0.983032	0.035	193
31	31	0.983031	0.035	193
32	32	0.983031	0.035	193
33	33	0.983031	0.035	193
34	34	0.983031	0.035	193
35	35	0.983031	0.035	193
36	36	0.983031	0.035	193
37	37	0.983031	0.035	193
38	38	0.983031	0.035	193

	step	f	gamma	k1
39	39	0.983031	0.035	193
40	40	0.983031	0.035	193
41	41	0.983031	0.035	193
42	42	0.983031	0.035	193
43	43	0.983031	0.035	193
44	44	0.983031	0.035	193
45	45	0.983031	0.035	193
46	46	0.983031	0.035	193
47	47	0.983031	0.035	193
48	48	0.983031	0.035	193

Ввод [40]:

```
import plotly.express as px

px.line(data_frame=df_10c1, x = 'step', y =  'gamma', title = 'Gamma chang
```



Ввод [41]:

```
px.line(data_frame=df_10c1, x = 'step', y =  'k1',
        title = 'Iteration number for each step for optimal alpha searchi
        log_y = True)
```

Ввод [42]:

```
px.line(data_frame=df_10c1, x = 'step', y =  'f', title = 'Function value
```

Ввод [43]:

```
x_arr[-1]
```

Out[43]:

```
array([-0.96544508,  0.99140524,  0.99785405,  0.99946341,  0.99986556,
        0.99996431,  0.99998363,  0.99996535,  0.99986965,  0.99948023])
```

Case 2: $\alpha^k = \text{argmin } f(x^k - \alpha \nabla f(x^k))$ (fminbound)

Ввод [44]:
```
1  _, _, _, df_10c2,x_arr = jax_gradient_descent(x, 0.5, 1e-6, adapt_gamma=Tr
```

```
[-1.5  1.   1.   1.   1.   1.   1.   1.   1.   1. ]
[0.98695921 1.39791347 1.         1.         1.         1.
 1.         1.         1.         1.        ]
[1.04435344 1.03909442 1.06157526 1.         1.         1.
 1.         1.         1.         1.        ]
[1.00933525 1.02379427 1.00487148 1.01468685 1.         1.
 1.         1.         1.         1.        ]
[1.00630153 1.00483777 1.00865283 1.00231948 1.00310984 1.
 1.         1.         1.         1.        ]
[1.00106857 1.00341911 1.00115765 1.0027979  1.0003767  1.00077724
 1.         1.         1.         1.        ]
[1.00086783 1.00073497 1.00143929 1.00054091 1.00080577 1.00014147
 1.00016871 1.         1.         1.        ]
[1.00016125 1.00052806 1.00021234 1.00052723 1.0001115  1.0002359
 1.00002314 1.00004266 1.         1.        ]
[1.00013352 1.00011954 1.00024569 1.00010862 1.00017464 1.00004645
 1.00006247 1.00000815 1.00000931 1.        ]
[1.00002625 1.00008685 1.00003866 1.00009814 1.00002569 1.00005652
 1.00000898 1.00001757 1.00000134 1.00000236]
[1.00002193 1.00002036 1.00004318 1.00002104 1.00003563 1.0000116
 1.00001683 1.00000351 1.00000445 1.00000252]
[1.00000448 1.00001492 1.00000709 1.00001835 1.00000547 1.0000124
 1.0000025  1.00000513 1.00000118 1.00000349]
[1.00000376 1.00000358 1.00000778 1.00000404 1.00000711 1.00000262
 1.00000401 1.00000117 1.00000197 1.00000356]
[1.00000079 1.00000265 1.00000132 1.00000346 1.00000112 1.00000262
 1.00000064 1.00000143 1.00000104 1.00000384]
[1.00000067 1.00000065 1.00000143 1.00000078 1.00000141 1.00000057
 1.00000093 1.00000047 1.00000122 1.00000385]
[1.00000014 1.00000048 1.00000025 1.00000066 1.00000023 1.00000055
 1.00000019 1.00000051 1.000001   1.00000392]
```

Ввод [45]:

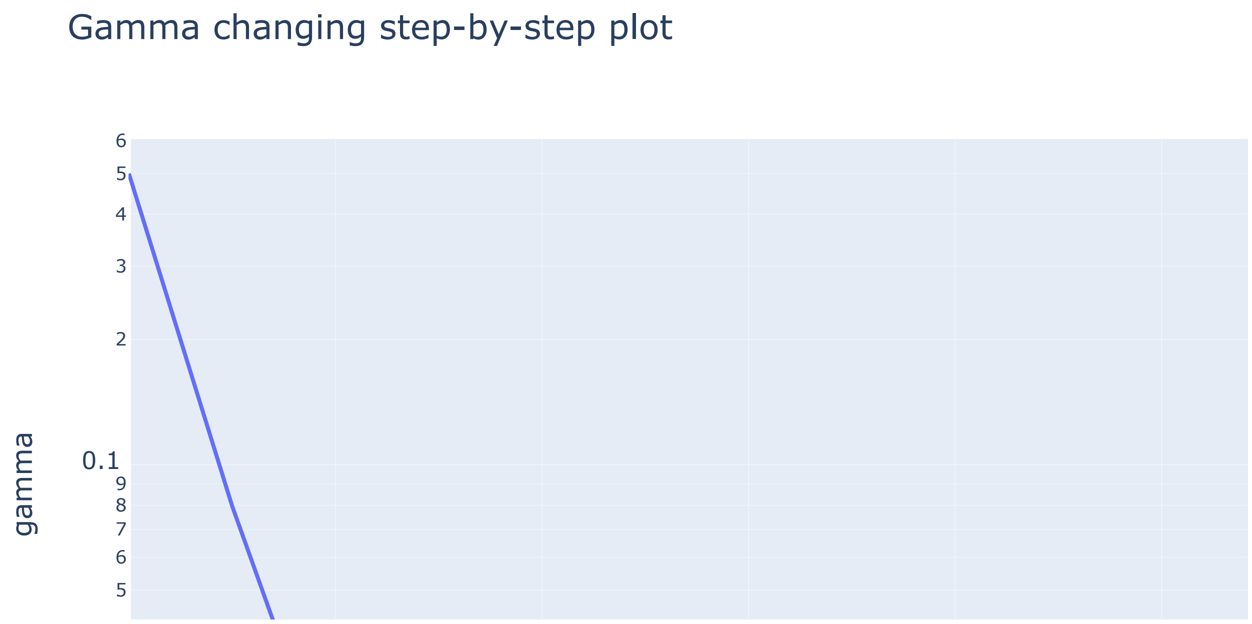
```
df_10c2
```

Out[45]:

	step	f	gamma	k1
0	0	7.812500e+00	0.500000	0
1	1	3.844007e+00	0.079583	16
2	2	9.476149e-02	0.016133	19
3	3	1.209657e-02	0.028924	12
4	4	1.789349e-03	0.026275	12
5	5	2.892026e-04	0.031192	11
6	6	4.915493e-05	0.027122	10
7	7	8.650979e-06	0.031606	9
8	8	1.563118e-06	0.027275	10
9	9	2.880018e-07	0.031701	9
10	10	5.389329e-08	0.027318	10
11	11	1.020021e-08	0.031738	9
12	12	1.947917e-09	0.027336	9
13	13	3.743375e-10	0.031762	8
14	14	7.228925e-11	0.027341	8
15	15	1.400672e-11	0.031779	7
16	16	2.720929e-12	0.027343	8

Ввод [46]:

```
px.line(data_frame=df_10c2, x = 'step', y =  'gamma', title = 'Gamma chang
```



Ввод [47]:

```
px.line(data_frame=df_10c2, x = 'step', y =  'k1',
        title = 'Iteration number for each step for optimal alpha searchi
        log_y = True)
```

Ввод [48]:

```
px.line(data_frame=df_10c2, x = 'step', y =  'f', title = 'Function value
```

Ввод [49]:

```
x_arr[-1]
```

Out[49]:

```
array([1.00000012, 1.00000012, 1.00000027, 1.00000015, 1.00000028,
       1.00000013, 1.00000024, 1.0000003 , 1.00000104, 1.00000393])
```

Case 3: $\alpha = 0.1$

Ввод [50]:

```
_, _, _, df_10c3,x_arr = jax_gradient_descent(x, 0.1, 1e-6, adapt_gamma=Fa
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:7: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars
```

Ввод [51]:

```
df_10c3
```

Out[51]:

	step	f
0	0	7.812500e+00
1	1	1.408301e+01
2	2	7.497906e+01
3	3	6.383094e+04
4	4	1.052642e+14
5	5	4.777553e+41
6	6	4.466594e+124
7	7	inf
8	8	inf
9	9	NaN

Ввод [52]:

```
x_arr
```

Out[52]:

```
[array([-1.5,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ]),
 array([1.625, 1.5  , 1.   , 1.   , 1.   , 1.   , 1.   , 1.   , 1.   ,
        1.   ]),
 array([-2.021875, -0.94375 ,  1.5     ,  1.      ,  1.      ,  1.      ,
         1.      ,  1.      ,  1.      ,  1.      ]),
 array([11.26284487,  0.13759414, -1.64373437,  1.5       ,  1.        ,
         1.        ,  1.        ,  1.        ,  1.        ,  1.        ]),
 array([-2.26493743e+03,  5.05757180e+01,  2.31094795e+00, -9.19254922e-01,
         1.50000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 1.85902991e+10,  1.84516239e+06,  1.00521282e+03,  6.05154600e-01,
        -1.66198816e+00,  1.50000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([-1.02796687e+31,  1.28188353e+20,  1.36022454e+12,  4.04180735e+05,
         2.63812020e+00, -8.95118148e-01,  1.50000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 1.73803026e+93,  3.88983574e+61,  6.56887486e+39,  7.40084215e+23,
         6.53448265e+10,  1.22506260e+00, -1.67950540e+00,  1.50000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([-8.40024555e+279,  1.11412941e+186,  6.04779368e+122,
         1.72600461e+079,  2.19089858e+047,  1.70797854e+021,
         3.27760880e+000, -8.71704644e-001,  1.50000000e+000,
         1.00000000e+000]),
 array([             nan,              nan,              nan,
         1.46303225e+245,  1.19163677e+158,  1.92001464e+094,
         1.16687628e+042,  2.71612489e+000, -1.69605241e+000,
         1.50000000e+000])]
```

Case 4: $\alpha = 0.5$

Ввод [53]:

```
_, _, _, df_10c4,x_arr = jax_gradient_descent(x, 0.5, 1e-6, adapt_gamma=Fa
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

invalid value encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars
```

Ввод [54]:

```
df_10c4
```

Out[54]:

	step	f
0	0	7.812500e+00
1	1	1.562042e+05
2	2	9.856757e+17
3	3	2.451558e+56
4	4	3.771956e+171
5	5	inf
6	6	inf
7	7	NaN

Ввод [55]:

```
x_arr
```

Out[55]:

```
[array([-1.5,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ]),
 array([14.125,  3.5   ,  1.    ,  1.    ,  1.    ,  1.    ,  1.    ,  1.    ,
         1.    ,  1.    ]),
 array([-2.22801719e+04,  8.30312500e+01,  2.35000000e+01,  1.00000000e+
00,
         1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+
00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 8.84800910e+13,  9.88240789e+08, -8.98476230e+04,  1.10350000e+
03,
         1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+
00,
         1.00000000e+00,  1.00000000e+00]),
 array([-5.54149146e+42,  7.93636837e+27,  1.95904214e+18,  5.39523582e+
09,
         2.43542350e+06,  1.00000000e+00,  1.00000000e+00,  1.00000000e+
00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 1.36135062e+129,  5.74172181e+085,  6.58238678e+055,
```

Case 5: $\alpha = 1$

Ввод [56]:

```
_, _, _, df_10c5,x_arr = jax_gradient_descent(x, 1, 1e-6, adapt_gamma=Fals
```

```
/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:6: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/377399078
6.py:7: RuntimeWarning:

overflow encountered in power

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:3: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

overflow encountered in double_scalars

/var/folders/fc/8vk24n953y36xydr3hkmr9y00000gn/T/ipykernel_71893/158174172
8.py:5: RuntimeWarning:

invalid value encountered in double_scalars
```

Ввод [57]:

```
df_10c5
```

Out[57]:

	step	f
0	0	7.812500e+00
1	1	3.113716e+06
2	2	1.240043e+23
3	3	7.810336e+72
4	4	1.951502e+222
5	5	inf
6	6	NaN

Ввод [58]:
```
x_arr
```

Out[58]:
```
[array([-1.5,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ]),
 array([29.75,  6.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,  1.   ,
         1.   ]),
 array([-4.19608375e+05,  1.72250000e+02,  1.41000000e+02,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 1.18209512e+18,  7.04203178e+11, -4.47307428e+07,  7.95210000e+04,
         1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([-2.64287887e+55,  1.94197830e+33,  3.41559295e+24, -4.23731059e+13,
         2.52943578e+10,  1.00000000e+00,  1.00000000e+00,  1.00000000e+00,
         1.00000000e+00,  1.00000000e+00]),
 array([ 2.95360056e+167,  2.79392350e+111, -6.37555940e+074,
         4.66651019e+049, -2.58927935e+032,  2.55921814e+021,
         1.00000000e+000,  1.00000000e+000,  1.00000000e+000,
         1.00000000e+000]),
 array([            -inf,              nan,  4.17765914e+225,
        -2.04178773e+143,  8.98827864e+099, -1.48770049e+061,
         2.61983899e+043,  1.00000000e+000,  1.00000000e+000,
         1.00000000e+000])]
```

How many iterations fminbound do

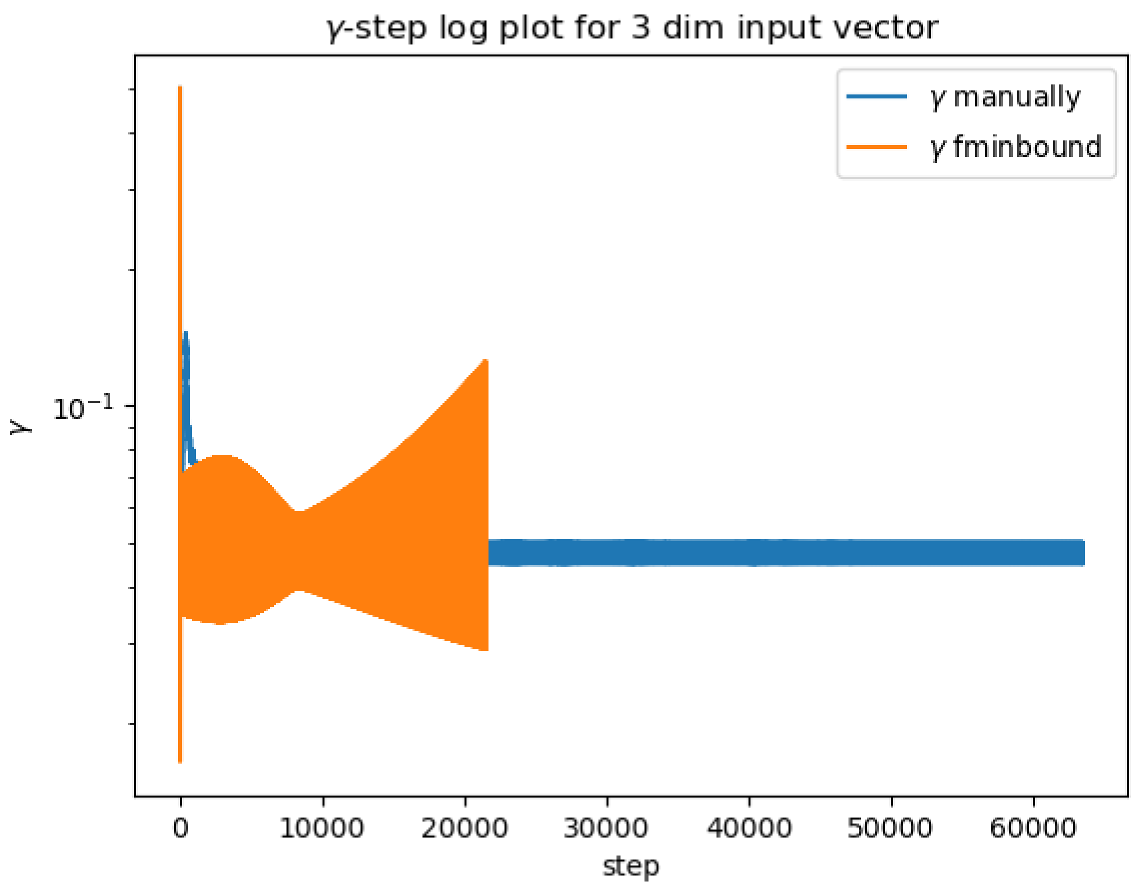
Ввод [59]:
```python
def gamma_adapt(gamma):
    z = task7_func_analyt(x - gamma*task7_grad_analyt(x))
    return z
gamma = fminbound(gamma_adapt, 0, 1,
                  xtol=1e-05, maxfun=100, disp=3)
```

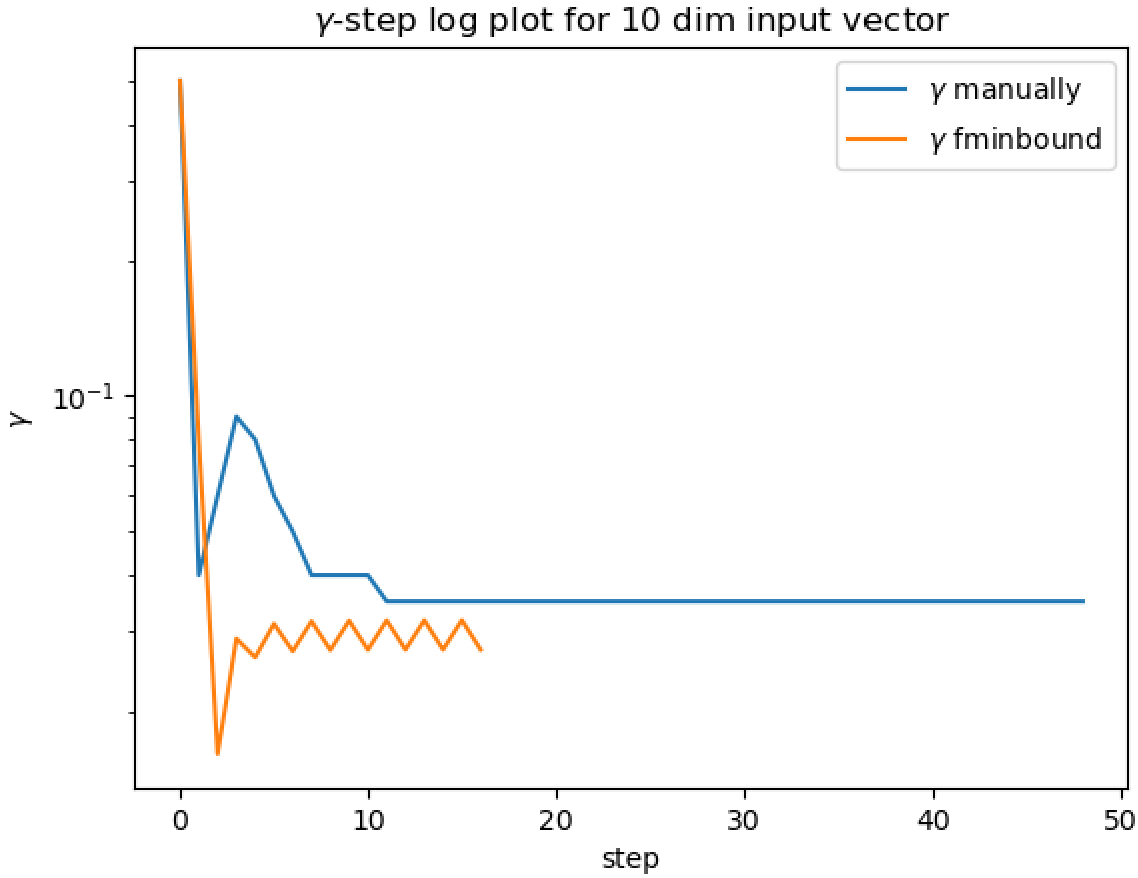
```
 Func-count     x          f(x)          Procedure
    1        0.381966     46010.8        initial
    2        0.618034      397401        golden
    3        0.236068     4405.23        golden
    4        0.263758     7758.98        parabolic
    5        0.206276     2149.54        parabolic
    6        0.127486     105.993        golden
    7        0.138594     192.638        parabolic
    8        0.118485     60.3616        parabolic
    9       0.0732276     4.27469        golden
   10       0.0870785     4.80913        parabolic
   11       0.0796484     3.84407        parabolic
   12       0.0787963     3.85209        parabolic
   13       0.0796769     3.84413        parabolic
   14       0.0795794     3.84401        parabolic
   15       0.0795827     3.84401        parabolic
   16        0.079586     3.84401        parabolic

Optimization terminated successfully:
```

Answer: in worst case - 16

Ввод [60]:

```
dfs3 = [df_3c1, df_3c2, df_3c3, df_3c4, df_3c5]
dfs10 = [df_10c1, df_10c2, df_10c3, df_10c4, df_10c5]
labels = ['$\gamma$ manually', '$\gamma$ fminbound', '$\gamma$ = 0.1',
          '$\gamma$ = 0.5', '$\gamma$ = 1']
for i in range(2):
    plt.plot(dfs3[i].step, dfs3[i].gamma, label=labels[i])
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('$\gamma$')
plt.title('$\gamma$-step log plot for 3 dim input vector')
plt.legend()
plt.show()

for i in range(2):
    plt.plot(dfs10[i].step, dfs10[i].gamma, label=labels[i])
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('$\gamma$')
plt.title('$\gamma$-step log plot for 10 dim input vector')
plt.legend()
plt.show()
```

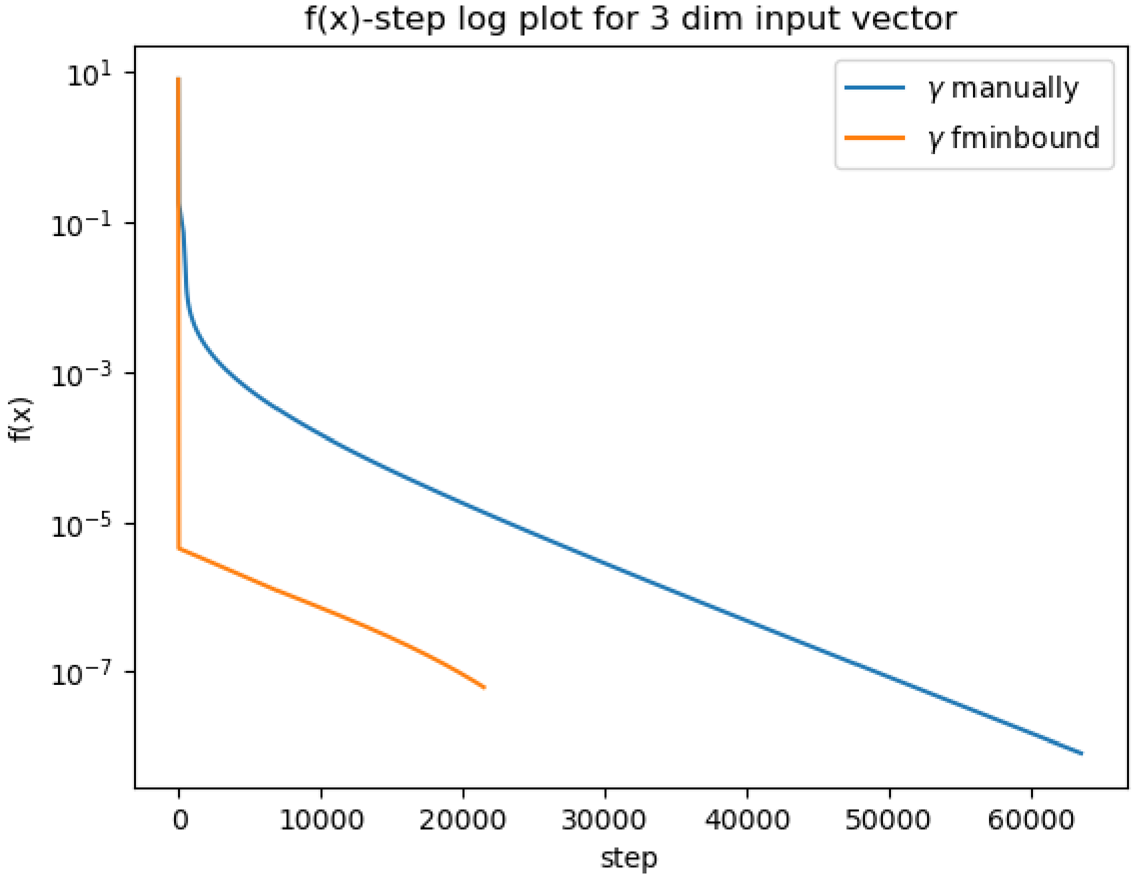


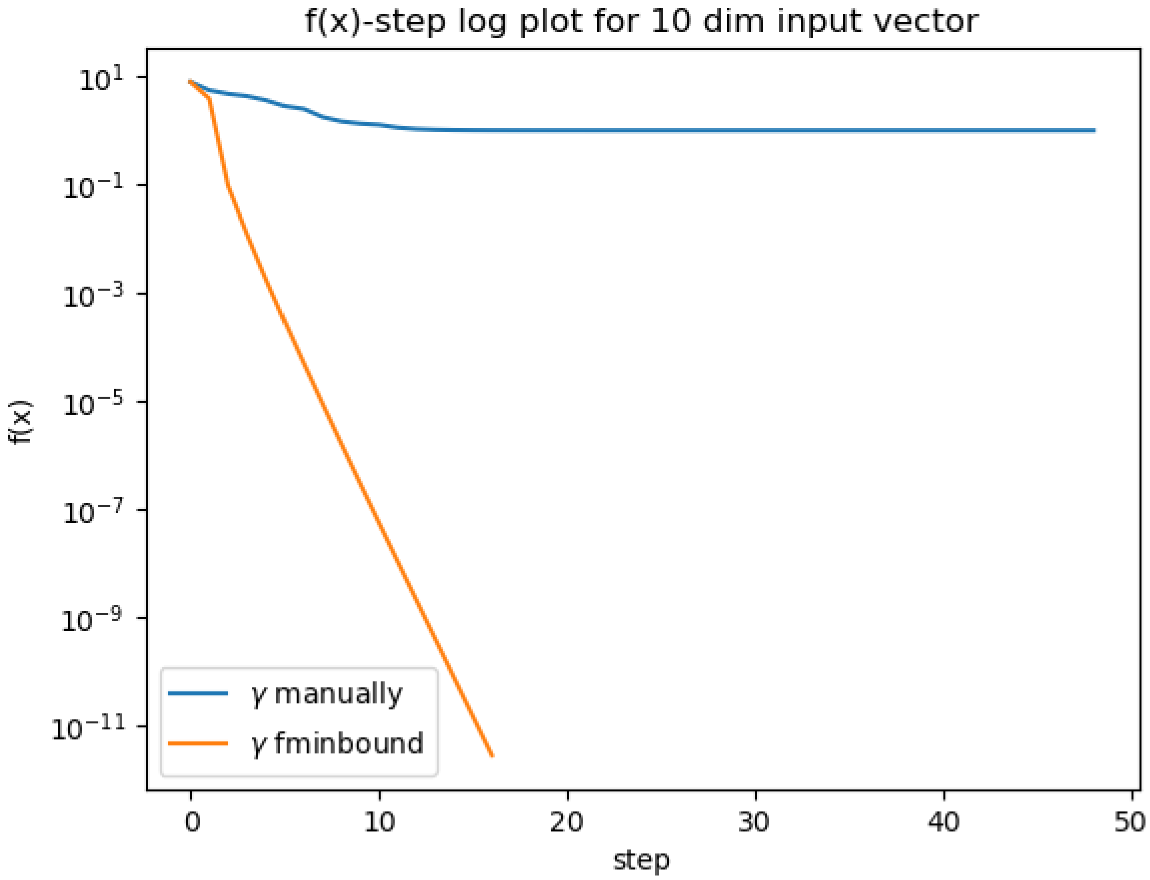


γ-step log plot for 10 dim input vector
γ manually
γ fminbound
10^{-1}
γ
0
10
20
30
40
50
step

Ввод [61]:

```
for i in range(2):
    plt.plot(dfs3[i].step, dfs3[i].f, label=labels[i])
# plt.xscale('log')
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('f(x)')
plt.title('f(x)-step log plot for 3 dim input vector')
plt.legend()
plt.show()

for i in range(2):
    plt.plot(dfs10[i].step, dfs10[i].f, label=labels[i])
# plt.xscale('log')
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('f(x)')
plt.title('f(x)-step log plot for 10 dim input vector')
plt.legend()
plt.show()
```

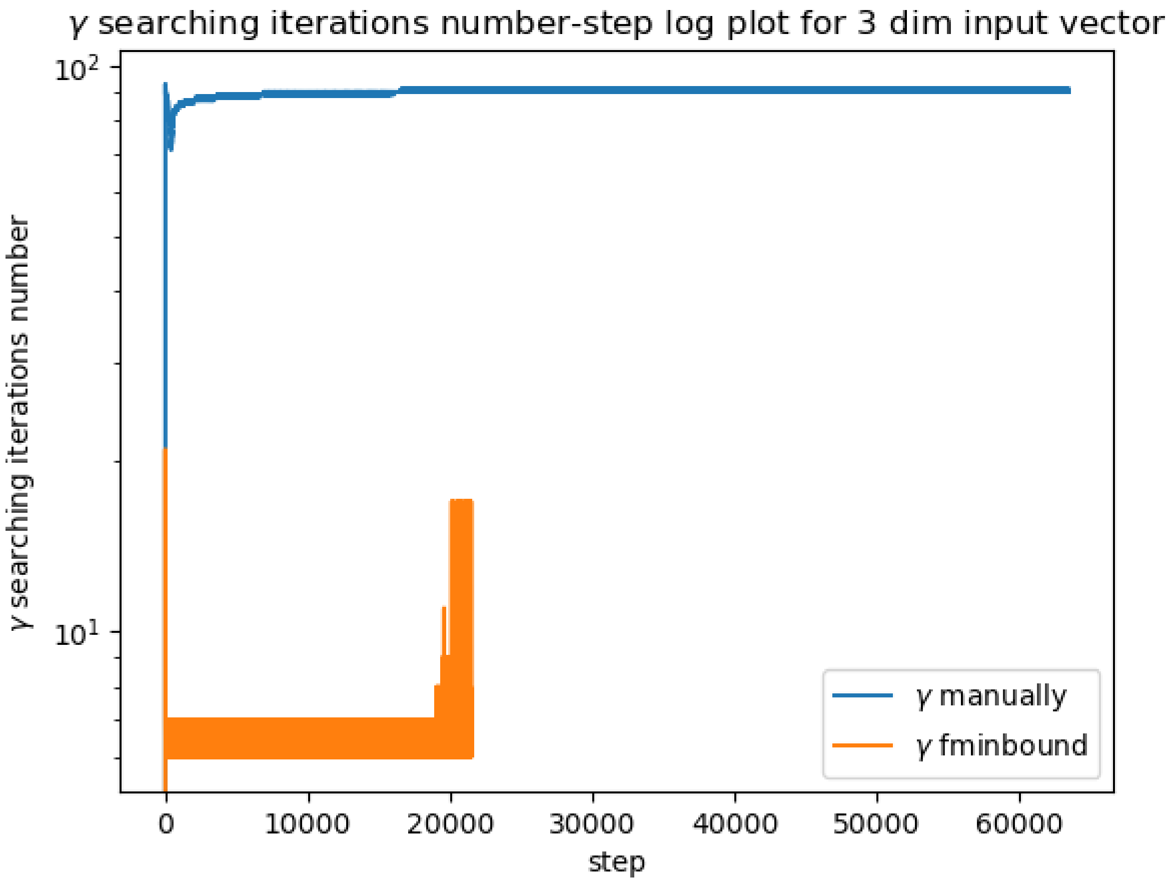


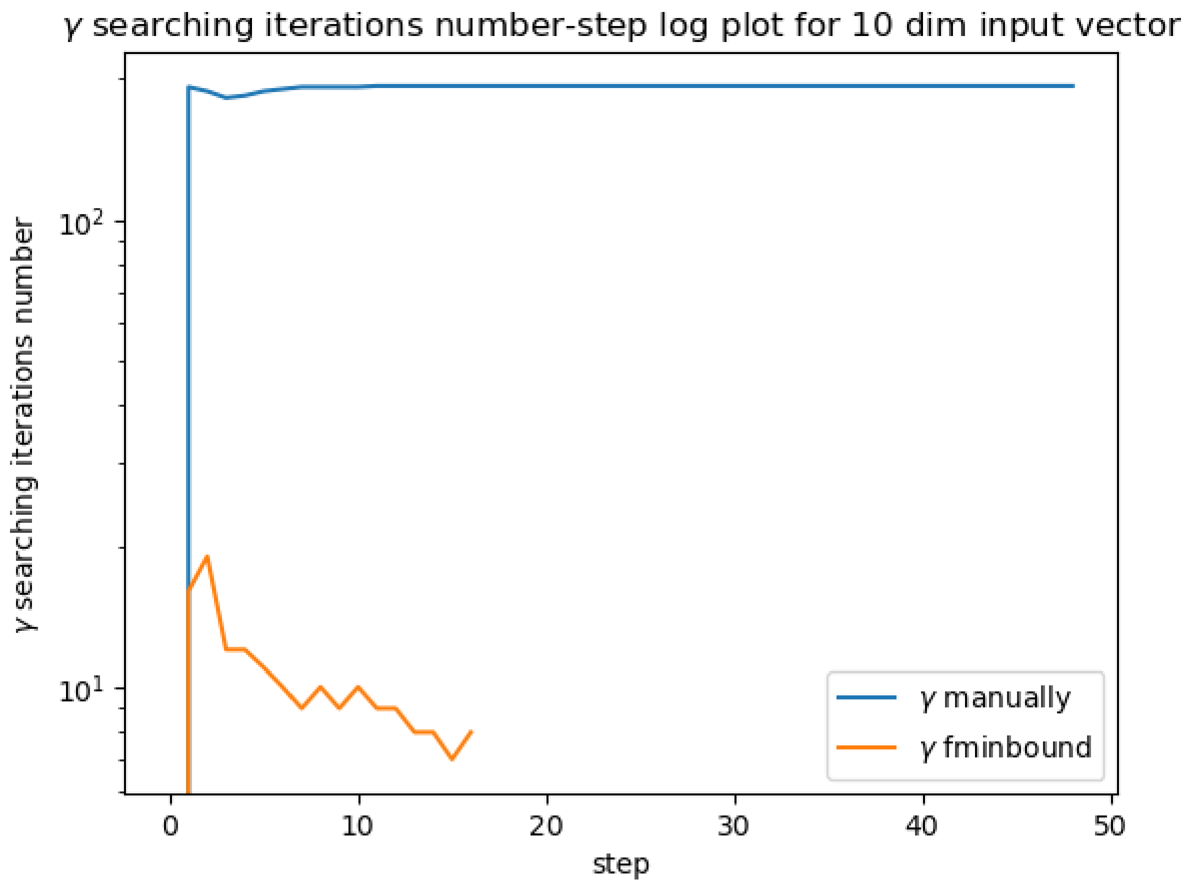


f(x)-step log plot for 10 dim input vector
10^{1}
10^{-1}
10^{-3}
10^{-5}
10^{-7}
10^{-9}
10^{-11}
f(x)
0
10
20
30
40
50
step
γ manually
γ fminbound

Ввод [62]:

```
for i in range(2):
    plt.plot(dfs3[i].step, dfs3[i].k1, label=labels[i])
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('$\gamma$ searching iterations number')
plt.title('$\gamma$ searching iterations number-step log plot for 3 dim in
plt.legend()
plt.show()

for i in range(2):
    plt.plot(dfs10[i].step, dfs10[i].k1, label=labels[i])
plt.yscale('log')
plt.xlabel('step')
plt.ylabel('$\gamma$ searching iterations number')
plt.title(
    '$\gamma$ searching iterations number-step log plot for 10 dim input v
plt.legend()
plt.show()
```





Report

According to this research, we can consider that

1. Steepest gradient descent with complicated function has bad convergence when Armiho condition used.
2. If built-in library smart fminbound searching applied, we have good convergence with high-dimension input data.
3. If built-in library smart fminbound searching applied, number of iteration of gradient descent can decreased in **2-3** times with ratio to Armiho condition.
4. Fminbound approach has bigger accuracy in huge-dimension case.
5. Also in this task I tried use different stopping condition like norm of gradient and norm of x^{k+1} and x^k differences.
6. The opinion exist (my opiniion) that the fact about better convergence on huge-dimension input data can be explained with closest norm to searching optimal poitn $\mathbf{x}^\star$, but it's not true because to x^0 for 3 and 10 dimension data spectral norm equal to 2.5