Минимизация функций

15	9.1.15, 9.4.5, 9.5.15, 9.6.15

9.1.15

Задача 9.1 Методом Ньютона найти минимум и максимум унимодальной на отрезке $[a,b]$ функции $f(x)$ с точностью $\varepsilon = 10^{-6}$. Предусмотреть подсчет числа итераций, потребовавшихся для достижения заданной точности.

9.1.15	$x^4 - e^x$	0	2

Необходимый теоретический материал

Направлением спуска является ньютоновское направление.

$\bar{x}^{(n)}$ может быть определена из необходимого условия экстремума:

$$g^{(n)} + G^{(n)}(\bar{x}^{(n)} - x^{(n)}) = 0.$$

Таким образом, чтобы попасть из точки $x^{(n)}$ в точку $\bar{x}^{(n)}$, нужно переместиться вдоль вектора $p^{(n)} = \bar{x}^{(n)} - x^{(n)}$, который определяется из системы линейных алгебраических уравнений

$$G^{(n)} p^{(n)} = -g^{(n)}. \qquad (10.30)$$

Вектор $p^{(n)}$ принято называть *ньютоновским направлением*, а метод спуска

$$x^{(n+1)} = x^{(n)} + \alpha_n p^{(n)} \qquad (10.31)$$

с таким выбором $p^{(n)}$ — *методом Ньютона*.

Отметим, что ньютоновское направление является направлением спуска. В самом деле, в силу равенства (10.30) для $p^{(n)}$ верна формула $p^{(n)} = -[G^{(n)}]^{-1} g^{(n)}$. Матрица $[G^{(n)}]^{-1}$ положительно определена (это следует из положительной определенности матрицы $G^{(n)}$). Поэтому

$$(f'(x^{(n)}), p^{(n)}) = -([G^{(n)}]^{-1} g^{(n)}, g^{(n)}) < 0.$$

Таким образом, условие (10.14) выполняется и $p^{(n)}$ действительно задает направление спуска.

квадратичная скорость сходимости позволяет определить простой критерий останова: |x_n - x_n+1| < eps

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


def f(x):
    return x**4 - np.exp(x)

def df(x):
    return 4*x**3 - np.exp(x)

def ddf(x):
    return 12*x**2 - np.exp(x)
```

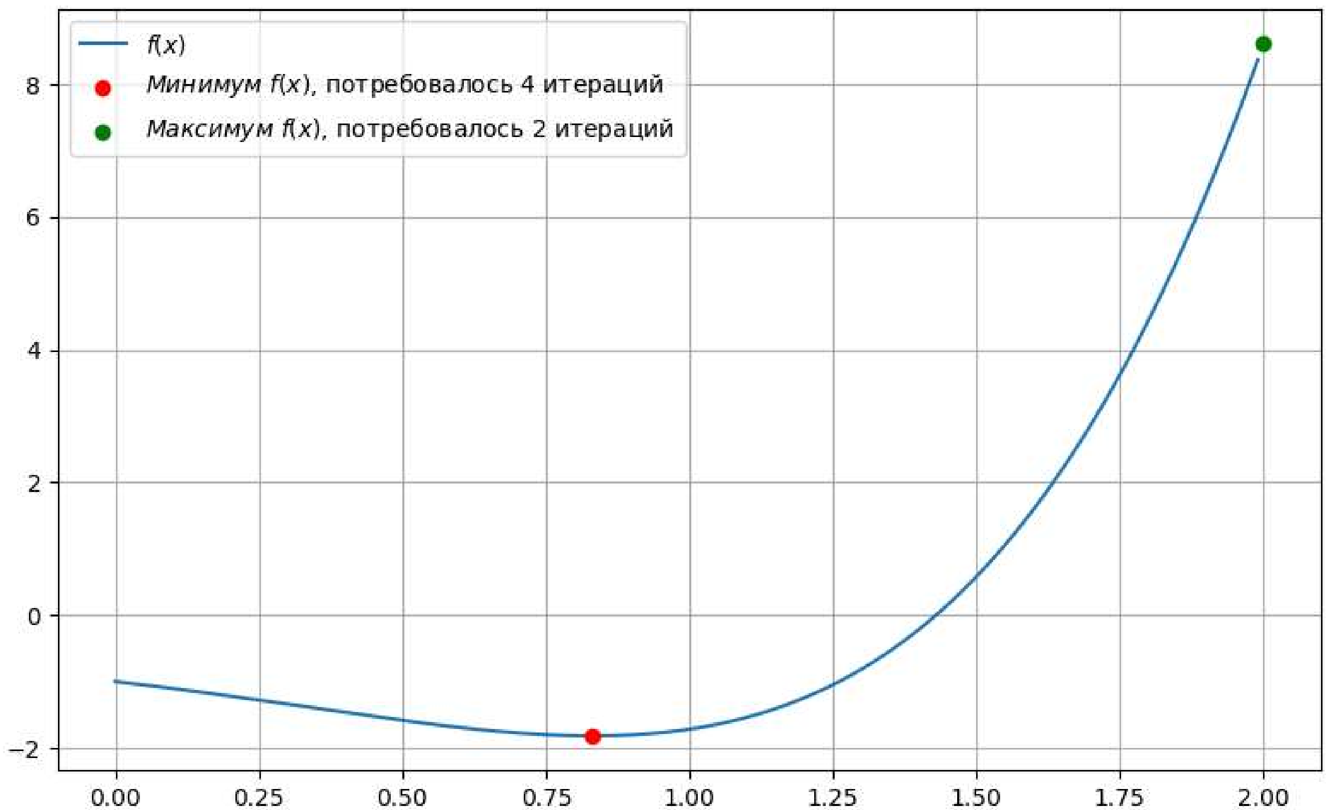
```
def newton_func(f, df, ddf, a, b, eps, start, extr = False):
    x = start
    i = 0

    while abs(df(x) / ddf(x)) > eps and a <= x <= b:
      step = df(x) / ddf(x)
      if extr == False:
        x -= step #min   Ньютоновское направление определяется ихз систе
      else:
        x += step
      i += 1
    x = min(b,max(a,x))
    return x, i

a, b = 0, 2
eps = 1e-6

start1 = (a+b)/2
x1, i1 = newton_func(f, df, ddf, a, b, eps, start1)
y1 = f(x1)
print(f"Минимум функции достигается в x = {x1}, f(x) = {y1}, за {i1} ите

start2 = 1.5
x2, i2 = newton_func(f, df, ddf, a, b, eps, start2, True)
y2 = f(x2)
print(f"Правый локальный максимум функции достигается в x = {x2}, f(x) =

# start3 = 0.25
# x3, i3 = newton_func(f, df, ddf, a, b, eps, start3)
# y3 = f(x3)
# print(f"Правый локальный максимум функции достигается в x = {x3}, f(x)
```

```
Минимум функции достигается в x = 0.8310314521178208, f(x) = -1.818738714288628, за 4 ит
Правый локальный максимум функции достигается в x = 2, f(x) = 8.61094390106935, за 2 ите
```

Вообще, метод Ньютона не подразумевает нахождение экстремумов, расположенных на концах рассматриваемого отрезка, но так как в условии это сказано, я это добавила.

В целом, для метода Ньютона очень важен выбор начального приближения

```
fig, ax = plt.subplots(figsize=(8, 5))
x_arr = np.arange(a, b, 0.01)

plt.plot(x_arr, f(x_arr), label='$f(x)$')
plt.scatter(x1, y1, color='red', label=f'$Минимум$ $f(x)$, потребовалос
plt.scatter(x2, y2, color='green', label=f'$Максимум$ $f(x)$, потребова
plt.tight_layout()
plt.legend()
plt.grid()
```



Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

9.4.5

Задача 9.4. Функция $f(x)$ представлена частичной суммой ряда $f(x)=\sum_{i=1}^{n} u_i(x)$.

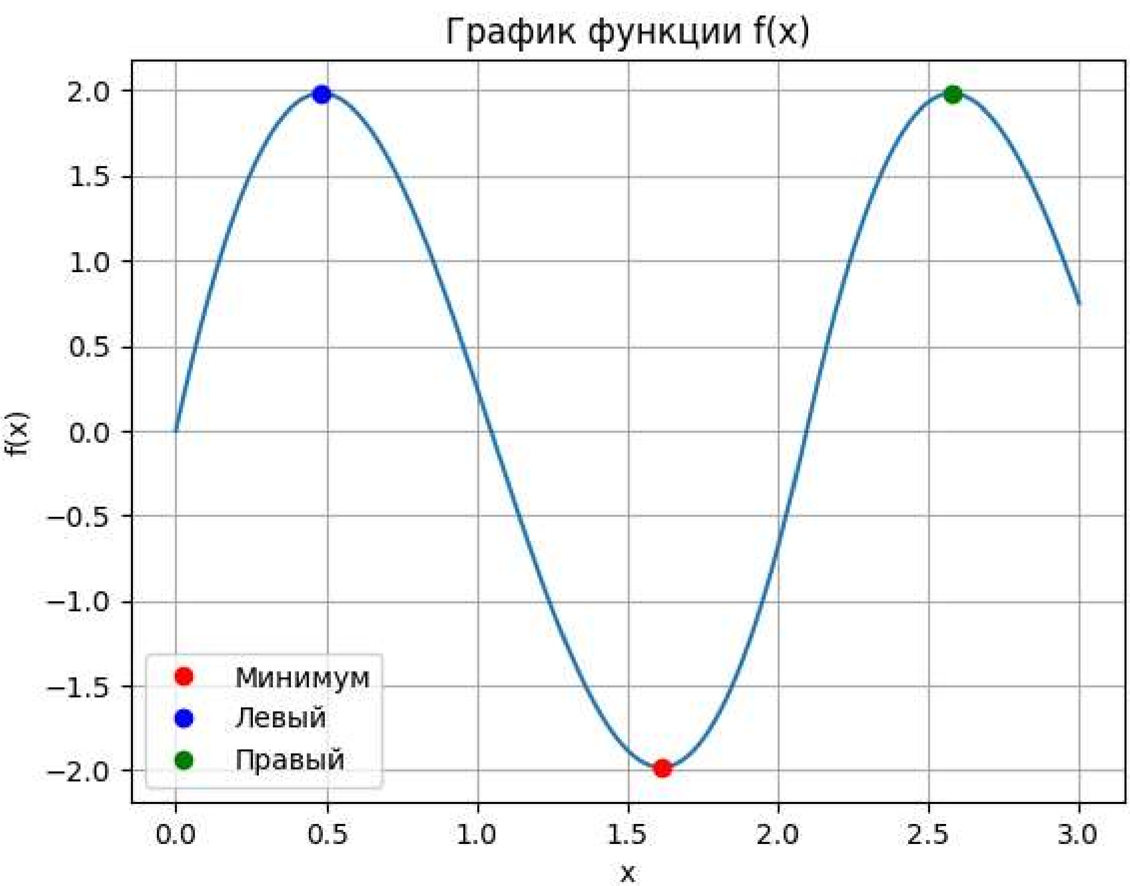
Построить график функции на заданном отрезке $[x_1, x_2]$ и найти ее минимумы и максимумы с указанной точностью ε.

№	$u_n(x)$	x_1	x_2	n	ε	Метод минимизации
9.4.5	$\dfrac{\sin(3nx)}{n^3-0.5}$	0	3	300	0.0001	Деления отрезка пополам

с. 247

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

# Параметры задачи
x1, x2 = 0, 3
n_terms = 300
epsilon = 0.0001

# Определение функции f(x)
def f(x):
    return sum(np.sin(3 * n * x) / (n**3 - 0.5) for n in range(1, n_ter

# Метод деления отрезка пополам для поиска минимума и максимума
def bisection_method(func, a, b, eps, find_min=True):
    while (b - a) / 2.0 > eps:
        midpoint = (a + b) / 2.0
        if find_min:
            if func(midpoint - eps) < func(midpoint + eps):
                b = midpoint
            else:
                a = midpoint
        else:
            if func(midpoint - eps) > func(midpoint + eps):
                b = midpoint
            else:
                a = midpoint
    return (a + b) / 2.0

# Поиск минимума функции f(x) на интервале [x1, x2]
xmin = bisection_method(f, x1, x2, epsilon, find_min=True)
fmin = f(xmin)

# Поиск максимума функции f(x) на интервале [x1, x2]
xmax = bisection_method(f, x1, x2, epsilon, find_min=False)
fmax = f(xmax)

# Поиск правого максимума функции f(x) на интервале [x1, x2]
xmax_r = bisection_method(f, 1.5, x2, epsilon, find_min=False)
fmax_r = f(xmax_r)

# Построение графика функции
x_values = np.linspace(x1, x2, 1000)
```

```
y_values = [f(x) for x in x_values]

plt.plot(x_values, y_values)
plt.plot(xmin, fmin, 'ro', label='Минимум')  # Точка минимума
plt.plot(xmax, fmax, 'bo', label='Левый')  # Точка максимума
plt.plot(xmax_r, fmax_r, 'go', label='Правый')  # Точка максимума
plt.title('График функции f(x)')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.legend()
plt.grid(True)
plt.show()

print(f'Минимум функции: x = {xmin}, f(x) = {fmin}')
print(f'Максимум функции: x = {xmax}, f(x) = {fmax}')
print(f'Максимум функции: x = {xmax_r}, f(x) = {fmax_r}')
```




```
Минимум функции: x = 1.611602783203125, f(x) = -1.9819354138791536
Максимум функции: x = 0.482757568359375, f(x) = 1.98193540946631
Максимум функции: x = 2.577117919921875, f(x) = 1.9819353850397754
```

9.5.15

Задача 9.5. Найти минимум функции 2-х переменных $f(x,y)$ с точностью $\varepsilon = 10^{-6}$ на прямоугольнике $[x_1, x_2] \times [y_1, y_2]$.

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Задать указанную в варианте функцию $f(x,y)$.
2. Построить графики функции и поверхностей уровня $f(x,y)$.
3. По графикам найти точки начального приближения к точкам экстремума.
4. Используя встроенные функции, найти экстремумы функции с заданной точностью (см. *ПРИЛОЖЕНИЕ 9.В*).

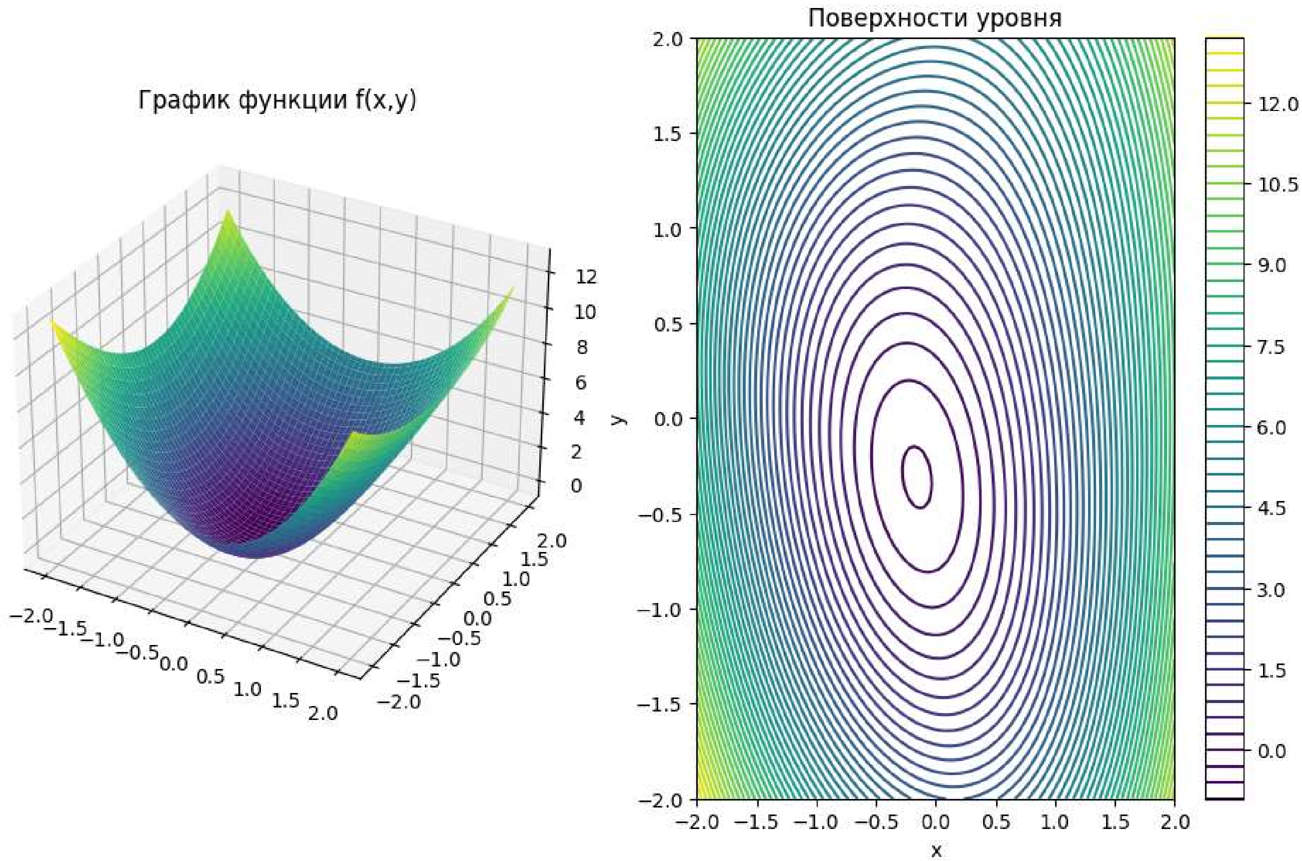
Таблица к задаче 9

№	f(x,y)	x_1	x_2	y_1	y_2
9.5.15	$2x^2 + y^2 + \cos(x + y - 2)$	-2	2	-2	2

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
from scipy.optimize import minimize


def f(x):
    return 2*x[0] ** 2 + x[1] ** 2 + np.cos(x[0] + x[1] - 2)

x1, x2 = -2, 2
y1, y2 = -2, 2
```

```
X = np.linspace(x1, x2, 100)
Y = np.linspace(y1, y2, 100)
X, Y = np.meshgrid(X, Y)
Z = f([X, Y])

fig = plt.figure(figsize=(12, 7))

ax1 = fig.add_subplot(121, projection='3d')
ax1.plot_surface(X, Y, Z, cmap='viridis')
ax1.set_title('График функции f(x,y)')

ax2 = fig.add_subplot(122)
contour = ax2.contour(X, Y, Z, 50)
ax2.set_title('Поверхности уровня')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
plt.colorbar(contour, ax=ax2)
# plt.axes()

plt.show()
```



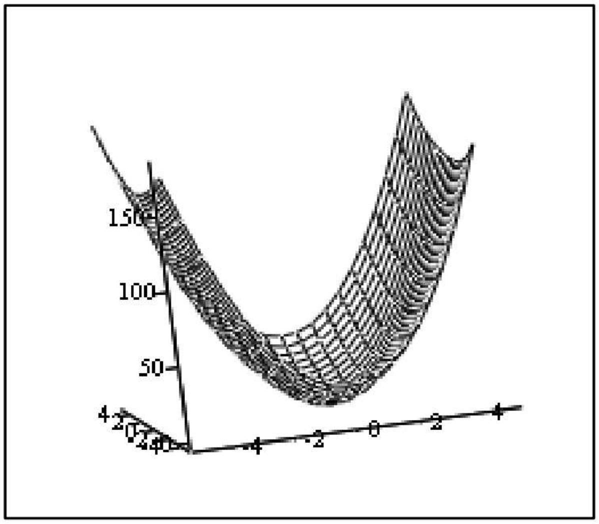

```
start = [0, 0] # по графику
result = minimize(f, start, bounds=[(-2, 2), (-2, 2)], tol=1e-6) # жесı

if result.success:
    print(f"Минимум функции достигается в точке x = {result.x[0]}, y = ·
else:
    print("Минимум функции не найден")
```

```
Минимум функции достигается в точке x = -0.155968181065988, y = -0.3119362462263531 со з
```

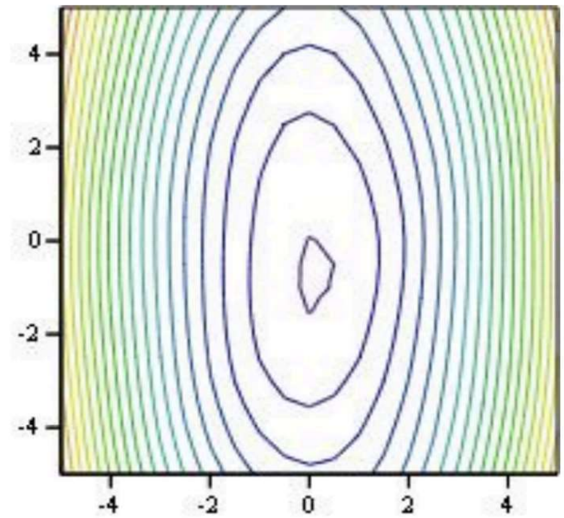
Задание функции 2-х переменных:

$$f(x, y) := 6\cdot(x - 0.3)^2 + (y + 0.4)^2 + 3\cdot(x - 0.3) + \sin(y - x)$$

График функции



Поверхности уровня функции



9.6.15

Задача 9.6. Указанным в индивидуальном варианте методом найти минимум квадратичной функции $f(x,y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y$ с точностью $\varepsilon = 10^{-6}$**.** Для решения задачи одномерной минимизации использовать метод Ньютона. Построить график функции f. Предусмотреть подсчет числа итераций, потребовавшихся для достижения заданной точности.

№	a_{11}	$2a_{12}$	a_{22}	$2a_{13}$	$2a_{23}$	Метод
9.6.15	0.5	-0.5	2.5	0	-9.5	Сопряженных направлений

Будем говорить, что ненулевые векторы $p^{(0)}, p^{(1)}, \ldots, p^{(m-1)}$ являются *взаимно сопряженными* (относительно матрицы A), если $(Ap^{(n)}, p^{(l)}) = 0$ для всех $n \neq l$.

Под *методом сопряженных направлений* для минимизации квадратичной функции (10.33) будем понимать метод

$$x^{(n+1)} = x^{(n)} + \alpha_n p^{(n)} \quad (n = 0, 1, 2, \ldots, m-1),$$

в котором направления $p^{(0)}, p^{(1)}, \ldots, p^{(m-1)}$ взаимно сопряжены, а шаги

$$\alpha_n = -\frac{(g^{(n)}, p^{(n)})}{(Ap^{(n)}, p^{(n)})}$$

получаются как решение задач одномерной минимизации:

$$\varphi_n(\alpha_n) = \min_{\alpha \geqslant 0} \varphi_n(\alpha), \quad \varphi_n(\alpha) = F(x^{(n)} + \alpha p^{(n)}).$$

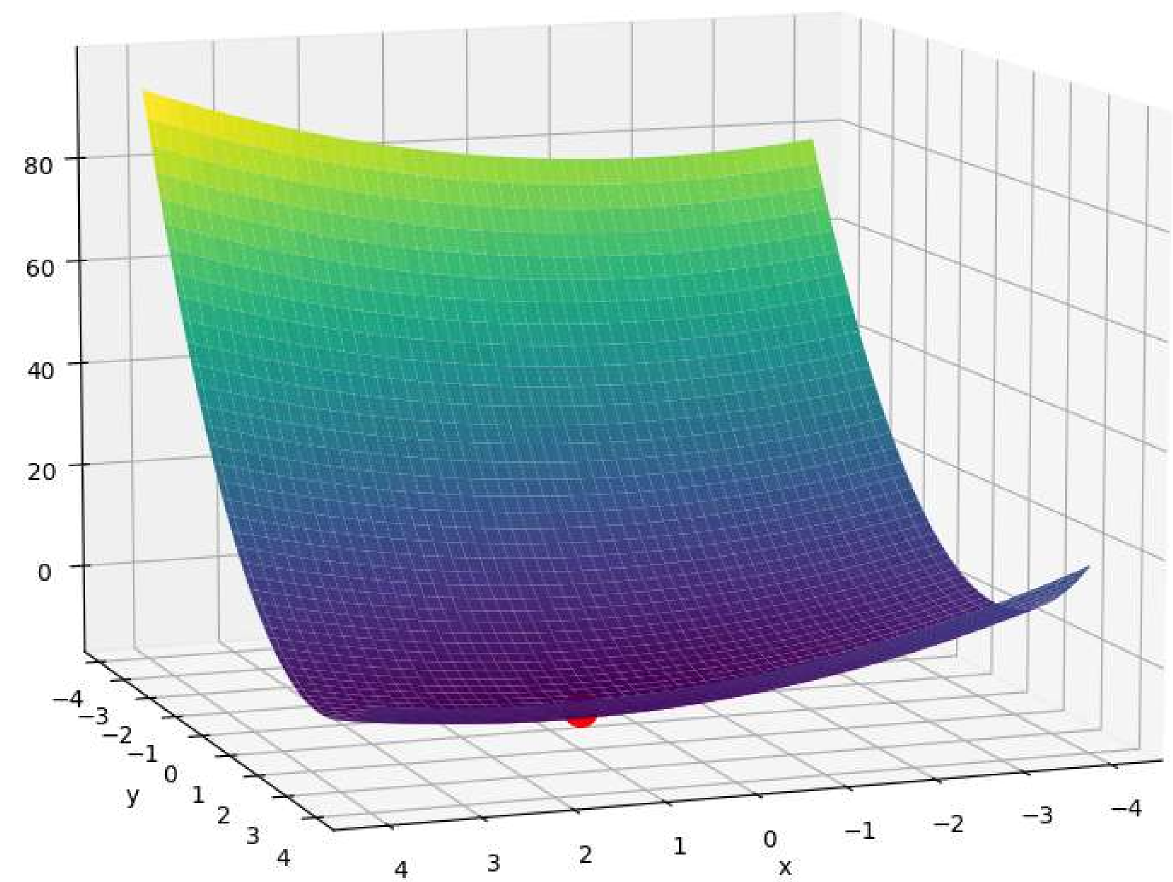
Теорема 10.4. *Метод сопряженных направлений позволяет найти точку минимума квадратичной функции* (10 33) *не более чем за m шагов.*

Методы сопряженных направлений отличаются один от другого способом построения сопряженных направлений. Наиболее известным среди них является *метод сопряженных градиентов*

```
import sympy as smp

# ЗАДАНИЕ 4
x = smp.symbols('x')  # x, вместо которого потом сможем подставлять каки
y = smp.symbols('y')  # y, вместо которого потом сможем подставлять каки
a11 = 0.5
a12 = -0.5
a22 = 2.5
a13 = 0
a23 = -9.5
f = a11 * (x ** 2) + a12 * x * y + a22 * (y ** 2) + a13 * x + a23 * y  #

# Определение функции f(x, y)
def func(x, y):
    return a11 * (x ** 2) + a12 * x * y + a22 * (y ** 2) + a13 * x + a2

x0 = 0  # первая координата начального приближения
y0 = 0  # вторая координата начального приближения
eps = 10 ** (-6)  # допустимая погрешность


def grad_1(f_grad, x_grad, y_grad):
    grad1_to_return = (f.diff(x)).subs({x: x_grad, y: y_grad})
    return grad1_to_return



def grad_2(f_grad, x_grad, y_grad):
    grad2_to_return = (f.diff(y)).subs({x: x_grad, y: y_grad})
    return grad2_to_return

def ConjugentGradients(f_this, x0_this, y0_this):
  print("\nМЕТОД: Метод сопряженных градиентов")

  i = 0
  l = smp.symbols('l')
  x_prev = x0_this
  y_prev = y0_this
  h_1 = (-1) * grad_1(f_this, x_prev, y_prev)
  h_2 = (-1) * grad_2(f_this, x_prev, y_prev)
  while (True):
    i = i + 1
    f_for_lambda = f_this.subs({x: x_prev + l * h_1, y: y_prev + l * h_
```

```
        f_for_lambda_diff = f_for_lambda.diff(l)
        this_lambda = smp.solvers.solvers.solve(f_for_lambda_diff, l)[0]
        x_next = x_prev + this_lambda * h_1
        y_next = y_prev + this_lambda * h_2
        beta = (grad_1(f_this, x_next, y_next) ** 2 + grad_2(f_this, x_next,
        h_1 = (-1) * grad_1(f_this, x_next, y_next) + beta * h_1
        h_2 = (-1) * grad_2(f_this, x_next, y_next) + beta * h_2
        x_prev = x_next
        y_prev = y_next
        # print("НОМЕР ИТЕРАЦИИ:", i, "\nТОЧКА: [", x_prev, ", ", y_prev, "
        if (max(abs(grad_1(f_this, x_prev, y_prev)), abs(grad_2(f_this, x_pr
          print("ИТОГОВОЕ КОЛИЧЕСТВО ИТЕРАЦИЙ: ", i, "\nТОЧКА ЧИСЛЕННОГО ГЛ
          break
    return x_prev, y_prev

x_root, y_root = ConjugentGradients(f, x0, y0)

x = np.linspace(-4, 4, 100)
y = np.linspace(-4, 4, 100)
x, y = np.meshgrid(x, y)
fig = plt.figure(figsize=(20, 10))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, a11 * (x ** 2) + a12 * x * y + a22 * (y ** 2) + a1
ax.view_init(10, 70)
ax.scatter(x_root, y_root, func(x_root, y_root) , color = 'red', s=200)
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```

```
МЕТОД: Метод сопряженных градиентов
ИТОГОВОЕ КОЛИЧЕСТВО ИТЕРАЦИЙ:  2
ТОЧКА ЧИСЛЕННОГО ГЛОБАЛЬНОГО МИНИМУМА: [ 1.00000000000000 ,  2.00000000000000 ]
```



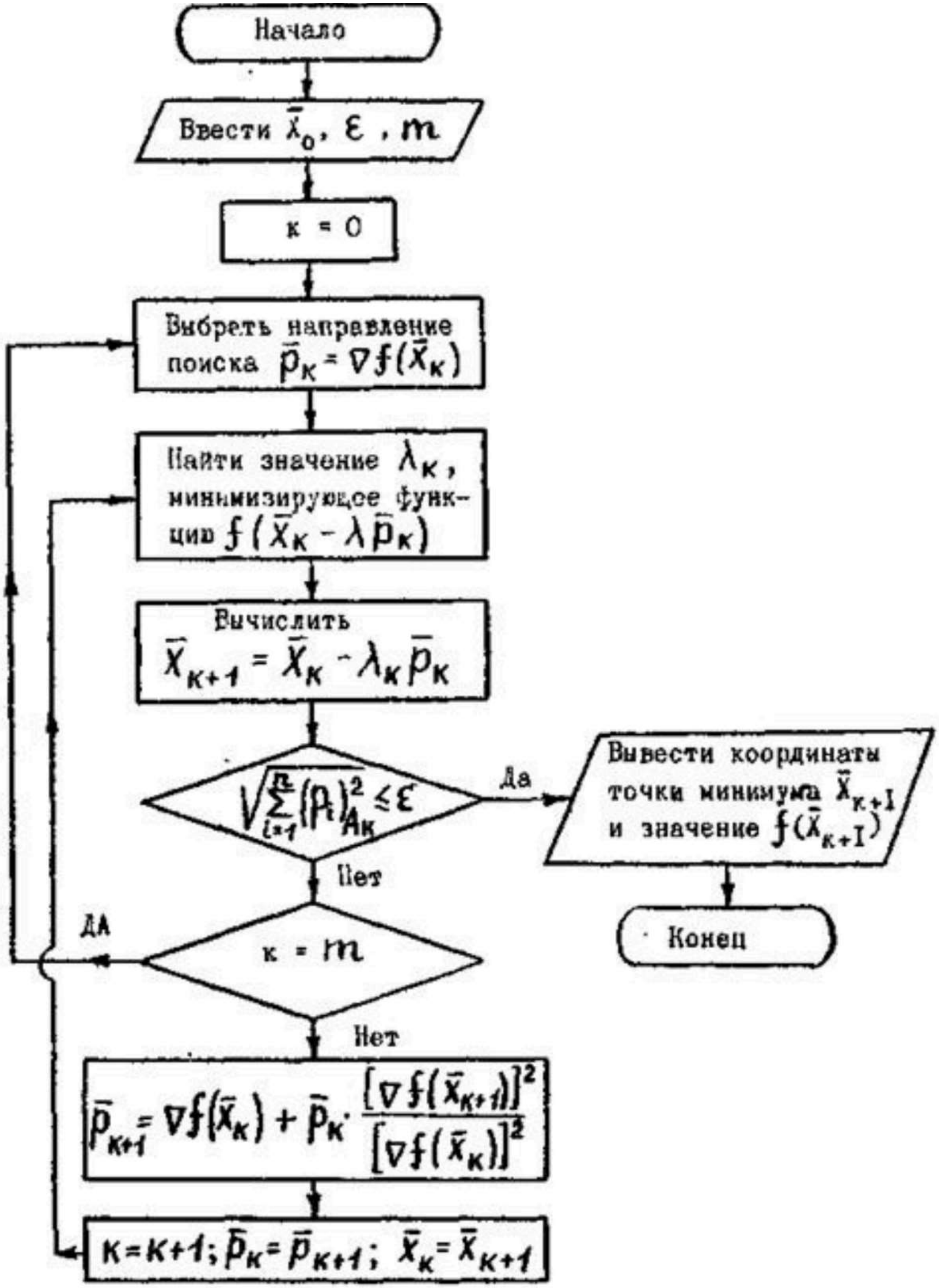


Рис. 12. Блок-схема метода сопряженных направлений

```
import numpy as np

# Коэффициенты функции
a11, a12, a22 = 0.5, -0.25, 2.5
a13, a23 = 0, -4.75

# Определение функции f(x, y)
def f(x, y):
    return a11*x**2 + 2*a12*x*y + a22*y**2 + 2*a13*x + 2*a23*y

# Матрица A и вектор b
A = np.array([
    [2*a11, 2*a12],
    [2*a12, 2*a22]], dtype=float)

b = np.array([2*a13, 2*a23], dtype=float)

def gradient_f(x, A, b):
    return A @ x + b

def conjugate_directions(A, b, x0, tol=1e-6, max_iter=1000):
```