|  |  |
| --- | --- |
| ***Вариант задания №2*** | |
| *Код программы:* | *Код оптимизированной программы:* |
| ***import numpy as np import matplotlib.pyplot as plt from mpl\_toolkits.mplot3d import Axes3D   # Определение целевой функции и функции ограничения def objective\_function(x, y):  return 10 \* (y - x) \*\* 2 + y \*\* 2   def constraint(x, y):  return x - (2 - y)   # Метод штрафных функций # Функция с штрафом: F(x,y) = f(x,y) + r\*g(x,y)^2, где g(x,y) - функция ограничения def penalty\_function(x, y, r):  return objective\_function(x, y) + r \* constraint(x, y) \*\* 2   # Начальные параметры x = 0.0 y = 0.0 r = 1.0 # начальный коэффициент штрафа alpha = 0.01 # шаг для градиентного спуска epsilon = 1e-6 # точность beta = 10.0 # множитель для увеличения штрафа max\_iterations = 10000 # максимальное число итераций max\_inner\_iterations = 100 # максимальное число внутренних итераций  # История для визуализации x\_history = [] y\_history = [] f\_history = []  # Внешний цикл - увеличение штрафа iteration = 0 converged = False  while not converged and iteration < max\_iterations:  # Внутренний цикл - поиск минимума текущей штрафной функции  inner\_iteration = 0  inner\_converged = False   while not inner\_converged and inner\_iteration < max\_inner\_iterations:  # Добавляем текущую точку в историю  x\_history.append(x)  y\_history.append(y)  f\_history.append(objective\_function(x, y))   # Расчет направления поиска (метод координатного спуска - 0-го порядка)  # Сначала делаем пробный шаг по x  x\_test = x + alpha  if penalty\_function(x\_test, y, r) < penalty\_function(x, y, r):  x = x\_test  else:  x\_test = x - alpha  if penalty\_function(x\_test, y, r) < penalty\_function(x, y, r):  x = x\_test   # Затем делаем пробный шаг по y  y\_test = y + alpha  if penalty\_function(x, y\_test, r) < penalty\_function(x, y, r):  y = y\_test  else:  y\_test = y - alpha  if penalty\_function(x, y\_test, r) < penalty\_function(x, y, r):  y = y\_test   # Проверяем условие сходимости внутреннего цикла  current\_value = penalty\_function(x, y, r)  next\_x = x + alpha if penalty\_function(x + alpha, y, r) < penalty\_function(x - alpha, y, r) else x - alpha  next\_y = y + alpha if penalty\_function(x, y + alpha, r) < penalty\_function(x, y - alpha, r) else y - alpha  next\_value = penalty\_function(next\_x, next\_y, r)   if abs(current\_value - next\_value) < epsilon:  inner\_converged = True   inner\_iteration += 1   # Проверка сходимости по ограничению  if abs(constraint(x, y)) < epsilon:  converged = True  else:  # Увеличиваем коэффициент штрафа  r \*= beta   iteration += 1  # Добавляем финальную точку в историю x\_history.append(x) y\_history.append(y) f\_history.append(objective\_function(x, y))  # Вывод результатов print(f"Найденная точка минимума: x = {x:.6f}, y = {y:.6f}") print(f"Значение функции: f(x,y) = {objective\_function(x, y):.6f}") print(f"Значение ограничения: g(x,y) = {constraint(x, y):.6f}") print(f"Количество итераций: {iteration}")  # Визуализация результатов # Создаем сетку для графика x\_range = np.linspace(-1, 3, 100) y\_range = np.linspace(-1, 3, 100) X, Y = np.meshgrid(x\_range, y\_range) Z = np.zeros\_like(X)  for i in range(len(x\_range)):  for j in range(len(y\_range)):  Z[j, i] = objective\_function(X[i, j], Y[j, i])  # Создаем 3D график fig = plt.figure(figsize=(14, 6))  # 3D поверхность функции ax1 = fig.add\_subplot(121, projection='3d') surf = ax1.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.8) ax1.set\_xlabel('X') ax1.set\_ylabel('Y') ax1.set\_zlabel('f(X,Y)') ax1.set\_title('Целевая функция')  # Отображаем точки оптимизации на поверхности path\_z = [objective\_function(x, y) for x, y in zip(x\_history, y\_history)] ax1.scatter(x\_history, y\_history, path\_z, color='r', s=10, alpha=0.6)  # 2D контурный график с ограничением ax2 = fig.add\_subplot(122) contour = ax2.contour(X, Y, Z, 50, cmap='viridis') ax2.set\_xlabel('X') ax2.set\_ylabel('Y') ax2.set\_title('Контурный график с ограничением')  # Отображаем ограничение x = 2 - y constraint\_x = np.linspace(-1, 3, 100) constraint\_y = 2 - constraint\_x ax2.plot(constraint\_x, constraint\_y, 'r--', label='x = 2 - y')  # Отображаем путь оптимизации ax2.plot(x\_history, y\_history, 'b-', alpha=0.6) ax2.scatter(x\_history, y\_history, color='b', s=10, alpha=0.6) ax2.scatter(x, y, color='r', s=100, marker='\*', label='Минимум')  ax2.legend() plt.colorbar(contour, ax=ax2) plt.tight\_layout() plt.show()*** | ***import numpy as np import matplotlib.pyplot as plt from mpl\_toolkits.mplot3d import Axes3D   def objective\_function(x, y):  """Целевая функция f(x, y) = 10\*(y - x)^2 + y^2"""  return 10 \* (y - x) \*\* 2 + y \*\* 2   def constraint(x, y):  """Ограничение x = 2 - y, преобразованное в g(x, y) = x - (2 - y)"""  return x - (2 - y)   def penalty\_function(x, y, r):  """Штрафная функция F(x,y) = f(x,y) + r\*g(x,y)^2"""  return objective\_function(x, y) + r \* constraint(x, y) \*\* 2   def coordinate\_descent\_step(x, y, r, alpha):  """Выполняет один шаг метода координатного спуска"""  # Шаг по x  if penalty\_function(x + alpha, y, r) < penalty\_function(x, y, r):  x = x + alpha  elif penalty\_function(x - alpha, y, r) < penalty\_function(x, y, r):  x = x - alpha   # Шаг по y  if penalty\_function(x, y + alpha, r) < penalty\_function(x, y, r):  y = y + alpha  elif penalty\_function(x, y - alpha, r) < penalty\_function(x, y, r):  y = y - alpha   return x, y   def check\_inner\_convergence(x, y, r, alpha, epsilon):  """Проверяет сходимость внутреннего цикла"""  current\_value = penalty\_function(x, y, r)  next\_x = x + alpha if penalty\_function(x + alpha, y, r) < penalty\_function(x - alpha, y, r) else x - alpha  next\_y = y + alpha if penalty\_function(x, y + alpha, r) < penalty\_function(x, y - alpha, r) else y - alpha  next\_value = penalty\_function(next\_x, next\_y, r)   return abs(current\_value - next\_value) < epsilon   def optimize(x0=0.0, y0=0.0, r0=1.0, alpha=0.01, epsilon=1e-6, beta=10.0,  max\_iterations=10000, max\_inner\_iterations=100):  """  Основная функция оптимизации с методом штрафных функций и координатным спуском   Параметры:  x0, y0 - начальная точка  r0 - начальный коэффициент штрафа  alpha - шаг для координатного спуска  epsilon - точность  beta - множитель для увеличения штрафа  max\_iterations - максимальное число внешних итераций  max\_inner\_iterations - максимальное число внутренних итераций   Возвращает:  x, y - найденная точка минимума  history - история точек для визуализации  iterations - число итераций  """  x, y = x0, y0  r = r0   # История для визуализации  history = {'x': [x], 'y': [y], 'f': [objective\_function(x, y)]}   # Внешний цикл - увеличение штрафа  iteration = 0  converged = False   while not converged and iteration < max\_iterations:  # Внутренний цикл - поиск минимума текущей штрафной функции  inner\_iteration = 0  inner\_converged = False   while not inner\_converged and inner\_iteration < max\_inner\_iterations:  # Шаг оптимизации  x, y = coordinate\_descent\_step(x, y, r, alpha)   # Записываем историю  history['x'].append(x)  history['y'].append(y)  history['f'].append(objective\_function(x, y))   # Проверяем условие сходимости внутреннего цикла  inner\_converged = check\_inner\_convergence(x, y, r, alpha, epsilon)  inner\_iteration += 1   # Проверка сходимости по ограничению  if abs(constraint(x, y)) < epsilon:  converged = True  else:  # Увеличиваем коэффициент штрафа  r \*= beta   iteration += 1   return x, y, history, iteration   def visualize\_results(x\_opt, y\_opt, history):  """Визуализирует результаты оптимизации"""  # Создаем сетку для графика  x\_range = np.linspace(-1, 3, 100)  y\_range = np.linspace(-1, 3, 100)  X, Y = np.meshgrid(x\_range, y\_range)  Z = np.zeros\_like(X)   for i in range(len(x\_range)):  for j in range(len(y\_range)):  Z[j, i] = objective\_function(X[i, j], Y[j, i])   # Создаем 3D график  fig = plt.figure(figsize=(14, 6))   # 3D поверхность функции  ax1 = fig.add\_subplot(121, projection='3d')  surf = ax1.plot\_surface(X, Y, Z, cmap='viridis', alpha=0.8)  ax1.set\_xlabel('X')  ax1.set\_ylabel('Y')  ax1.set\_zlabel('f(X,Y)')  ax1.set\_title('Целевая функция')   # Отображаем точки оптимизации на поверхности  path\_z = [objective\_function(x, y) for x, y in zip(history['x'], history['y'])]  ax1.scatter(history['x'], history['y'], path\_z, color='r', s=10, alpha=0.6)   # 2D контурный график с ограничением  ax2 = fig.add\_subplot(122)  contour = ax2.contour(X, Y, Z, 50, cmap='viridis')  ax2.set\_xlabel('X')  ax2.set\_ylabel('Y')  ax2.set\_title('Контурный график с ограничением')   # Отображаем ограничение x = 2 - y  constraint\_x = np.linspace(-1, 3, 100)  constraint\_y = 2 - constraint\_x  ax2.plot(constraint\_x, constraint\_y, 'r--', label='x = 2 - y')   # Отображаем путь оптимизации  ax2.plot(history['x'], history['y'], 'b-', alpha=0.6)  ax2.scatter(history['x'], history['y'], color='b', s=10, alpha=0.6)  ax2.scatter(x\_opt, y\_opt, color='r', s=100, marker='\*', label='Минимум')   ax2.legend()  plt.colorbar(contour, ax=ax2)  plt.tight\_layout()  plt.show()   def main():  # Параметры оптимизации  x0, y0 = 0.0, 0.0 # начальная точка  r0 = 1.0 # начальный коэффициент штрафа  alpha = 0.01 # шаг для координатного спуска  epsilon = 1e-6 # точность  beta = 10.0 # множитель для увеличения штрафа   # Запуск оптимизации  x\_opt, y\_opt, history, iterations = optimize(x0, y0, r0, alpha, epsilon, beta)   # Вывод результатов  print(f"Найденная точка минимума: x = {x\_opt:.6f}, y = {y\_opt:.6f}")  print(f"Значение функции: f(x,y) = {objective\_function(x\_opt, y\_opt):.6f}")  print(f"Значение ограничения: g(x,y) = {constraint(x\_opt, y\_opt):.6f}")  print(f"Количество итераций: {iterations}")   # Визуализация  visualize\_results(x\_opt, y\_opt, history)   if \_\_name\_\_ == "\_\_main\_\_":  main()*** |
| *Расчет метрики:* | *Расчет метрики:* |
|  | |