|  |  |
| --- | --- |
| ***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 def objective\_function(x, y):  return 10 \* (y - x) \*\* 2 + y \*\* 2   # Функция ограничения преобразованная так, чтобы g(x,y) > 0 # Ограничение x = 2 - y преобразуем в g(x,y) = (2 - y) - x # Тогда g(x,y) > 0 означает x < 2 - y def constraint(x, y):  return (2 - y) - x   # Барьерная функция, которая стремится к бесконечности при приближении к границе допустимой области def barrier\_function(x, y, mu):  constr\_val = constraint(x, y)  # Проверка, что мы находимся в допустимой области  if constr\_val <= 0:  return float('inf') # За пределами допустимой области возвращаем бесконечность  return objective\_function(x, y) - mu \* np.log(constr\_val)   # Начальные параметры # Начальная точка должна быть внутри допустимой области (g(x,y) > 0) x = 0.0 y = 0.0 mu = 1.0 # начальный параметр барьера alpha = 0.01 # шаг для координатного спуска epsilon = 1e-6 # точность mu\_reduction\_factor = 0.1 # множитель для уменьшения параметра барьера max\_iterations = 1000 # максимальное число итераций max\_inner\_iterations = 100 # максимальное число внутренних итераций  # История для визуализации x\_history = [] y\_history = [] f\_history = []  # Проверка, что начальная точка находится в допустимой области if constraint(x, y) <= 0:  # Если начальная точка не подходит, находим подходящую точку  for test\_x in np.linspace(-1, 1, 20):  for test\_y in np.linspace(-1, 1, 20):  if constraint(test\_x, test\_y) > 0:  x = test\_x  y = test\_y  break  if constraint(x, y) > 0:  break  # Внешний цикл - уменьшение параметра барьера iteration = 0 converged = False  while not converged and iteration < max\_iterations:  # Внутренний цикл - поиск минимума текущей барьерной 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))   # Текущее значение барьерной функции  current\_value = barrier\_function(x, y, mu)   # Расчет направления поиска (метод координатного спуска - 0-го порядка)  # Сначала делаем пробный шаг по x  x\_plus = x + alpha  x\_minus = x - alpha   # Проверяем, что пробные точки находятся в допустимой области  value\_plus\_x = barrier\_function(x\_plus, y, mu)  value\_minus\_x = barrier\_function(x\_minus, y, mu)   # Выбираем лучшее направление по x  if value\_plus\_x < current\_value and value\_plus\_x < value\_minus\_x:  x = x\_plus  elif value\_minus\_x < current\_value:  x = x\_minus   # Обновляем текущее значение после шага по x  current\_value = barrier\_function(x, y, mu)   # Затем делаем пробный шаг по y  y\_plus = y + alpha  y\_minus = y - alpha   # Проверяем, что пробные точки находятся в допустимой области  value\_plus\_y = barrier\_function(x, y\_plus, mu)  value\_minus\_y = barrier\_function(x, y\_minus, mu)   # Выбираем лучшее направление по y  if value\_plus\_y < current\_value and value\_plus\_y < value\_minus\_y:  y = y\_plus  elif value\_minus\_y < current\_value:  y = y\_minus   # Проверяем условие сходимости внутреннего цикла  new\_value = barrier\_function(x, y, mu)  if abs(current\_value - new\_value) < epsilon:  inner\_converged = True   inner\_iteration += 1   # Проверка условия сходимости внешнего цикла  if mu < epsilon:  converged = True  else:  # Уменьшаем параметр барьера  mu \*= mu\_reduction\_factor   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')  # Заштриховываем недопустимую область (x > 2 - y) invalid\_x = np.linspace(-1, 3, 100) invalid\_y = np.linspace(-1, 3, 100) invalid\_X, invalid\_Y = np.meshgrid(invalid\_x, invalid\_y) invalid\_mask = invalid\_X > (2 - invalid\_Y) ax2.contourf(invalid\_X, invalid\_Y, invalid\_mask.astype(float), levels=[0.5, 1.5], colors='gray', alpha=0.3)  # 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):  """  Функция ограничения преобразованная так, чтобы g(x,y) > 0  Ограничение x = 2 - y преобразуем в g(x,y) = (2 - y) - x  Тогда g(x,y) > 0 означает x < 2 - y  """  return (2 - y) - x   def barrier\_function(x, y, mu):  """  Барьерная функция, добавляющая штраф при приближении к границе допустимой области  Принимает:  x, y - координаты точки  mu - параметр барьера  Возвращает:  Значение барьерной функции или бесконечность, если точка недопустима  """  constr\_val = constraint(x, y)  if constr\_val <= 0:  return float('inf') # За пределами допустимой области возвращаем бесконечность  return objective\_function(x, y) - mu \* np.log(constr\_val)   def find\_feasible\_point():  """Находит начальную точку в допустимой области"""  for test\_x in np.linspace(-1, 1, 20):  for test\_y in np.linspace(-1, 1, 20):  if constraint(test\_x, test\_y) > 0:  return test\_x, test\_y  # Если не удалось найти точку, возвращаем безопасное значение  return 0.0, 1.0 # Точка (0,1) явно удовлетворяет ограничению x < 2 - y   def coordinate\_descent\_step(x, y, mu, alpha):  """  Выполняет один шаг метода координатного спуска для барьерной функции  Принимает:  x, y - текущая точка  mu - параметр барьера  alpha - шаг для координатного спуска  Возвращает:  Новые значения x, y  """  # Текущее значение барьерной функции  current\_value = barrier\_function(x, y, mu)   # Расчет направления поиска по x  x\_plus = x + alpha  x\_minus = x - alpha   value\_plus\_x = barrier\_function(x\_plus, y, mu)  value\_minus\_x = barrier\_function(x\_minus, y, mu)   # Выбираем лучшее направление по x  if value\_plus\_x < current\_value and value\_plus\_x < value\_minus\_x:  x = x\_plus  elif value\_minus\_x < current\_value:  x = x\_minus   # Обновляем текущее значение после шага по x  current\_value = barrier\_function(x, y, mu)   # Расчет направления поиска по y  y\_plus = y + alpha  y\_minus = y - alpha   value\_plus\_y = barrier\_function(x, y\_plus, mu)  value\_minus\_y = barrier\_function(x, y\_minus, mu)   # Выбираем лучшее направление по y  if value\_plus\_y < current\_value and value\_plus\_y < value\_minus\_y:  y = y\_plus  elif value\_minus\_y < current\_value:  y = y\_minus   return x, y   def check\_inner\_convergence(x, y, mu, prev\_value, epsilon):  """  Проверяет сходимость внутреннего цикла  Принимает:  x, y - текущая точка  mu - параметр барьера  prev\_value - предыдущее значение функции  epsilon - точность  Возвращает:  True, если достигнута сходимость, иначе False  """  new\_value = barrier\_function(x, y, mu)  return abs(prev\_value - new\_value) < epsilon   def optimize(x0=None, y0=None, mu0=1.0, alpha=0.01, epsilon=1e-6, mu\_reduction\_factor=0.1,  max\_iterations=1000, max\_inner\_iterations=100):  """  Основная функция оптимизации с методом барьерных функций и координатным спуском   Параметры:  x0, y0 - начальная точка (если None, то будет найдена допустимая точка)  mu0 - начальный параметр барьера  alpha - шаг для координатного спуска  epsilon - точность  mu\_reduction\_factor - множитель для уменьшения параметра барьера  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  """  # Если начальная точка не задана или находится вне допустимой области, находим подходящую  if x0 is None or y0 is None or constraint(x0, y0) <= 0:  x, y = find\_feasible\_point()  else:  x, y = x0, y0   mu = mu0   # История для визуализации  history = {'x': [x], 'y': [y], 'f': [objective\_function(x, y)]}   # Внешний цикл - уменьшение параметра барьера  iteration = 0  converged = False   while not converged and iteration < max\_iterations:  # Внутренний цикл - поиск минимума текущей барьерной функции  inner\_iteration = 0  inner\_converged = False  prev\_value = barrier\_function(x, y, mu)   while not inner\_converged and inner\_iteration < max\_inner\_iterations:  # Шаг оптимизации  x, y = coordinate\_descent\_step(x, y, mu, 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, mu, prev\_value, epsilon)  prev\_value = barrier\_function(x, y, mu)   inner\_iteration += 1   # Проверка условия сходимости внешнего цикла  if mu < epsilon:  converged = True  else:  # Уменьшаем параметр барьера  mu \*= mu\_reduction\_factor   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')   # Заштриховываем недопустимую область (x > 2 - y)  invalid\_x = np.linspace(-1, 3, 100)  invalid\_y = np.linspace(-1, 3, 100)  invalid\_X, invalid\_Y = np.meshgrid(invalid\_x, invalid\_y)  invalid\_mask = invalid\_X > (2 - invalid\_Y)  ax2.contourf(invalid\_X, invalid\_Y, invalid\_mask.astype(float), levels=[0.5, 1.5], colors='gray', alpha=0.3)   # 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  mu0 = 1.0 # начальный параметр барьера  alpha = 0.01 # шаг для координатного спуска  epsilon = 1e-6 # точность  mu\_reduction\_factor = 0.1 # множитель для уменьшения параметра барьера   # Запуск оптимизации (начальная точка будет найдена автоматически)  x\_opt, y\_opt, history, iterations = optimize(  mu0=mu0, alpha=alpha, epsilon=epsilon,  mu\_reduction\_factor=mu\_reduction\_factor  )   # 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:* |
|  | |