|  |  |
| --- | --- |
| ***Вариант задания №2*** | |
| *Код программы:* | *Код оптимизированной программы:* |
| ***import numpy as np import matplotlib.pyplot as plt from mpl\_toolkits.mplot3d import Axes3D from matplotlib import cm   def f(x, y):  return (1 / (1 + ((x - 2) / 3) \*\* 2 + ((y - 2) / 3) \*\* 2)) + (3 / (1 + (x - 1) \*\* 2 + ((y - 1) / 2) \*\* 2))   def df\_dx(x, y, h=0.0001):  return (f(x + h, y) - f(x - h, y)) / (2 \* h)   def df\_dy(x, y, h=0.0001):  return (f(x, y + h) - f(x, y - h)) / (2 \* h)   def d2f\_dx2(x, y, h=0.0001):  return (f(x + h, y) - 2 \* f(x, y) + f(x - h, y)) / (h \*\* 2)   def d2f\_dy2(x, y, h=0.0001):  return (f(x, y + h) - 2 \* f(x, y) + f(x, y - h)) / (h \*\* 2)   def d2f\_dxdy(x, y, h=0.0001):  return (f(x + h, y + h) - f(x + h, y - h) - f(x - h, y + h) + f(x - h, y - h)) / (4 \* h \*\* 2)   # Метод Гаусса (координатный спуск) def gauss\_method(start\_x, start\_y, alpha=0.1, max\_iterations=1000, tolerance=1e-6):  x = start\_x  y = start\_y  iteration = 0  convergence\_history = []   while iteration < max\_iterations:  old\_x = x  old\_y = y  old\_value = f(x, y)  convergence\_history.append((x, y, old\_value))   # Поиск по x  x\_step = alpha  if f(x + x\_step, y) < f(x, y):  x\_step = -alpha   while f(x + x\_step, y) > f(x, y):  x = x + x\_step   # Поиск по y  y\_step = alpha  if f(x, y + y\_step) < f(x, y):  y\_step = -alpha   while f(x, y + y\_step) > f(x, y):  y = y + y\_step   new\_value = f(x, y)   # Проверка сходимости  if abs(new\_value - old\_value) < tolerance:  break   iteration += 1   return x, y, f(x, y), convergence\_history   # Метод Ньютона def newton\_method(start\_x, start\_y, max\_iterations=100, tolerance=1e-6):  x = start\_x  y = start\_y  iteration = 0  convergence\_history = []   while iteration < max\_iterations:  old\_value = f(x, y)  convergence\_history.append((x, y, old\_value))   # Вычисление градиента и матрицы Гессе  gradient\_x = df\_dx(x, y)  gradient\_y = df\_dy(x, y)   hessian\_xx = d2f\_dx2(x, y)  hessian\_yy = d2f\_dy2(x, y)  hessian\_xy = d2f\_dxdy(x, y)   # Определитель матрицы Гессе  det\_hessian = hessian\_xx \* hessian\_yy - hessian\_xy \*\* 2   # Для поиска максимума матрица Гессе должна быть отрицательно определена  if det\_hessian == 0:  # Если определитель равен нулю, используем небольшой шаг по градиенту  x = x + 0.01 \* gradient\_x  y = y + 0.01 \* gradient\_y  else:  # Обновляем координаты с использованием метода Ньютона  delta\_x = (hessian\_yy \* gradient\_x - hessian\_xy \* gradient\_y) / det\_hessian  delta\_y = (hessian\_xx \* gradient\_y - hessian\_xy \* gradient\_x) / det\_hessian   # Для поиска максимума используем противоположное направление  x = x + delta\_x  y = y + delta\_y   new\_value = f(x, y)   # Проверка сходимости  if abs(new\_value - old\_value) < tolerance:  break   iteration += 1   return x, y, f(x, y), convergence\_history   # Визуализация функции def plot\_function\_and\_path(history, title):  # Создаем сетку  x = np.linspace(-1, 5, 100)  y = np.linspace(-1, 5, 100)  X, Y = np.meshgrid(x, y)  Z = np.zeros\_like(X)   # Вычисляем значения функции  for i in range(X.shape[0]):  for j in range(X.shape[1]):  Z[i, j] = f(X[i, j], Y[i, j])   # Создаем 3D график  fig = plt.figure(figsize=(12, 10))  ax1 = fig.add\_subplot(221, projection='3d')  surf = ax1.plot\_surface(X, Y, Z, cmap=cm.coolwarm, alpha=0.8)  ax1.set\_xlabel('X')  ax1.set\_ylabel('Y')  ax1.set\_zlabel('f(X, Y)')  ax1.set\_title(f'3D поверхность функции с траекторией ({title})')   # Добавляем путь оптимизации на 3D графике  path\_x = [p[0] for p in history]  path\_y = [p[1] for p in history]  path\_z = [p[2] for p in history]  ax1.plot(path\_x, path\_y, path\_z, 'r-o', lw=2, markersize=3)   # Создаем 2D график с контурами  ax2 = fig.add\_subplot(222)  contour = ax2.contourf(X, Y, Z, levels=50, cmap=cm.coolwarm)  fig.colorbar(contour, ax=ax2)  ax2.set\_xlabel('X')  ax2.set\_ylabel('Y')  ax2.set\_title(f'Контурный график с траекторией ({title})')   # Добавляем путь оптимизации на контурном графике  ax2.plot(path\_x, path\_y, 'r-o', lw=2, markersize=3)   # График сходимости  ax3 = fig.add\_subplot(212)  ax3.plot(range(len(path\_z)), path\_z, 'b-', lw=2)  ax3.set\_xlabel('Итерация')  ax3.set\_ylabel('f(X, Y)')  ax3.set\_title(f'График сходимости ({title})')  ax3.grid(True)   plt.tight\_layout()  return fig   # Основной блок def main():  # Начальная точка  start\_x = 0.0  start\_y = 0.0   # Запуск метода Гаусса  gauss\_x, gauss\_y, gauss\_value, gauss\_history = gauss\_method(start\_x, start\_y)  print("\nРезультаты метода Гаусса:")  print(f"Точка максимума: x = {gauss\_x:.6f}, y = {gauss\_y:.6f}")  print(f"Значение функции: f(x,y) = {gauss\_value:.6f}")  print(f"Количество итераций: {len(gauss\_history)}")   # Визуализация для метода Гаусса  gauss\_fig = plot\_function\_and\_path(gauss\_history, "Метод Гаусса")   # Запуск метода Ньютона  newton\_x, newton\_y, newton\_value, newton\_history = newton\_method(start\_x, start\_y)  print("\nРезультаты метода Ньютона:")  print(f"Точка максимума: x = {newton\_x:.6f}, y = {newton\_y:.6f}")  print(f"Значение функции: f(x,y) = {newton\_value:.6f}")  print(f"Количество итераций: {len(newton\_history)}")   # Визуализация для метода Ньютона  newton\_fig = plot\_function\_and\_path(newton\_history, "Метод Ньютона")   plt.show()   if \_\_name\_\_ == "\_\_main\_\_":  main()*** | ***import numpy as np import matplotlib.pyplot as plt from mpl\_toolkits.mplot3d import Axes3D from matplotlib import cm import time   class FunctionOptimizer:  def \_\_init\_\_(self, func, start\_x=0.0, start\_y=0.0, tolerance=1e-6):  """  Инициализация оптимизатора функции.   Parameters:  - func: оптимизируемая функция двух переменных  - start\_x, start\_y: начальные координаты  - tolerance: критерий остановки оптимизации  """  self.func = func  self.start\_x = start\_x  self.start\_y = start\_y  self.tolerance = tolerance   def \_compute\_gradient(self, x, y, h=0.0001):  """Вычисление градиента функции"""  df\_dx = (self.func(x + h, y) - self.func(x - h, y)) / (2 \* h)  df\_dy = (self.func(x, y + h) - self.func(x, y - h)) / (2 \* h)  return df\_dx, df\_dy   def \_compute\_hessian(self, x, y, h=0.0001):  """Вычисление матрицы Гессе"""  d2f\_dx2 = (self.func(x + h, y) - 2 \* self.func(x, y) + self.func(x - h, y)) / (h \*\* 2)  d2f\_dy2 = (self.func(x, y + h) - 2 \* self.func(x, y) + self.func(x, y - h)) / (h \*\* 2)  d2f\_dxdy = (self.func(x + h, y + h) - self.func(x + h, y - h) -  self.func(x - h, y + h) + self.func(x - h, y - h)) / (4 \* h \*\* 2)   return d2f\_dx2, d2f\_dy2, d2f\_dxdy   def gauss\_method(self, alpha=0.1, max\_iterations=1000):  """  Метод Гаусса (координатный спуск) для поиска максимума функции.   Parameters:  - alpha: начальный шаг  - max\_iterations: максимальное число итераций   Returns:  - x, y: найденные координаты максимума  - value: значение функции в точке максимума  - history: история оптимизации  - time\_elapsed: время выполнения алгоритма  """  start\_time = time.time()  x, y = self.start\_x, self.start\_y  iteration = 0  history = []   while iteration < max\_iterations:  old\_value = self.func(x, y)  history.append((x, y, old\_value))   # Поиск по x  x\_step = alpha  if self.func(x + x\_step, y) < self.func(x, y):  x\_step = -alpha   # Линейный поиск по x  while self.func(x + x\_step, y) > self.func(x, y):  x = x + x\_step   # Поиск по y  y\_step = alpha  if self.func(x, y + y\_step) < self.func(x, y):  y\_step = -alpha   # Линейный поиск по y  while self.func(x, y + y\_step) > self.func(x, y):  y = y + y\_step   new\_value = self.func(x, y)   # Проверка сходимости  if abs(new\_value - old\_value) < self.tolerance:  break   iteration += 1   time\_elapsed = time.time() - start\_time  return x, y, self.func(x, y), history, time\_elapsed   def newton\_method(self, max\_iterations=100):  """  Метод Ньютона для поиска максимума функции.   Parameters:  - max\_iterations: максимальное число итераций   Returns:  - x, y: найденные координаты максимума  - value: значение функции в точке максимума  - history: история оптимизации  - time\_elapsed: время выполнения алгоритма  """  start\_time = time.time()  x, y = self.start\_x, self.start\_y  iteration = 0  history = []   while iteration < max\_iterations:  old\_value = self.func(x, y)  history.append((x, y, old\_value))   # Вычисление градиента и матрицы Гессе  gradient\_x, gradient\_y = self.\_compute\_gradient(x, y)  hessian\_xx, hessian\_yy, hessian\_xy = self.\_compute\_hessian(x, y)   # Определитель матрицы Гессе  det\_hessian = hessian\_xx \* hessian\_yy - hessian\_xy \*\* 2   # Для поиска максимума матрица Гессе должна быть отрицательно определена  if abs(det\_hessian) < 1e-10:  # Если определитель близок к нулю, используем небольшой шаг по градиенту  x = x + 0.01 \* gradient\_x  y = y + 0.01 \* gradient\_y  else:  # Обновляем координаты с использованием метода Ньютона  delta\_x = (hessian\_yy \* gradient\_x - hessian\_xy \* gradient\_y) / det\_hessian  delta\_y = (hessian\_xx \* gradient\_y - hessian\_xy \* gradient\_x) / det\_hessian   # Для поиска максимума используем противоположное направление  x = x + delta\_x  y = y + delta\_y   new\_value = self.func(x, y)   # Проверка сходимости  if abs(new\_value - old\_value) < self.tolerance:  break   iteration += 1   time\_elapsed = time.time() - start\_time  return x, y, self.func(x, y), history, time\_elapsed   def visualize\_optimization(func, history, title):  """  Визуализация функции и пути оптимизации.   Parameters:  - func: оптимизируемая функция  - history: история оптимизации  - title: заголовок графика   Returns:  - fig: фигура matplotlib  """  # Создаем сетку  x = np.linspace(-1, 5, 100)  y = np.linspace(-1, 5, 100)  X, Y = np.meshgrid(x, y)  Z = np.zeros\_like(X)   # Вычисляем значения функции  for i in range(X.shape[0]):  for j in range(X.shape[1]):  Z[i, j] = func(X[i, j], Y[i, j])   # Создаем 3D график  fig = plt.figure(figsize=(12, 10))  ax1 = fig.add\_subplot(221, projection='3d')  surf = ax1.plot\_surface(X, Y, Z, cmap=cm.coolwarm, alpha=0.8)  ax1.set\_xlabel('X')  ax1.set\_ylabel('Y')  ax1.set\_zlabel('f(X, Y)')  ax1.set\_title(f'3D поверхность функции с траекторией ({title})')   # Добавляем путь оптимизации на 3D графике  path\_x = [p[0] for p in history]  path\_y = [p[1] for p in history]  path\_z = [p[2] for p in history]  ax1.plot(path\_x, path\_y, path\_z, 'r-o', lw=2, markersize=3)   # Создаем 2D график с контурами  ax2 = fig.add\_subplot(222)  contour = ax2.contourf(X, Y, Z, levels=50, cmap=cm.coolwarm)  fig.colorbar(contour, ax=ax2)  ax2.set\_xlabel('X')  ax2.set\_ylabel('Y')  ax2.set\_title(f'Контурный график с траекторией ({title})')   # Добавляем путь оптимизации на контурном графике  ax2.plot(path\_x, path\_y, 'r-o', lw=2, markersize=3)   # График сходимости  ax3 = fig.add\_subplot(212)  ax3.plot(range(len(path\_z)), path\_z, 'b-', lw=2)  ax3.set\_xlabel('Итерация')  ax3.set\_ylabel('f(X, Y)')  ax3.set\_title(f'График сходимости ({title})')  ax3.grid(True)   plt.tight\_layout()  return fig   def main():  """Основная функция программы"""   # Определение целевой функции  def target\_function(x, y):  return (1 / (1 + ((x - 2) / 3) \*\* 2 + ((y - 2) / 3) \*\* 2)) + (3 / (1 + (x - 1) \*\* 2 + ((y - 1) / 2) \*\* 2))   # Начальная точка  start\_x, start\_y = 0.0, 0.0   # Создание оптимизатора  optimizer = FunctionOptimizer(target\_function, start\_x, start\_y)   # Запуск метода Гаусса  gauss\_x, gauss\_y, gauss\_value, gauss\_history, gauss\_time = optimizer.gauss\_method()  print("\nРезультаты метода Гаусса:")  print(f"Точка максимума: x = {gauss\_x:.6f}, y = {gauss\_y:.6f}")  print(f"Значение функции: f(x,y) = {gauss\_value:.6f}")  print(f"Количество итераций: {len(gauss\_history)}")  print(f"Время выполнения: {gauss\_time:.6f} секунд")   # Визуализация для метода Гаусса  gauss\_fig = visualize\_optimization(target\_function, gauss\_history, "Метод Гаусса")   # Запуск метода Ньютона  newton\_x, newton\_y, newton\_value, newton\_history, newton\_time = optimizer.newton\_method()  print("\nРезультаты метода Ньютона:")  print(f"Точка максимума: x = {newton\_x:.6f}, y = {newton\_y:.6f}")  print(f"Значение функции: f(x,y) = {newton\_value:.6f}")  print(f"Количество итераций: {len(newton\_history)}")  print(f"Время выполнения: {newton\_time:.6f} секунд")   # Визуализация для метода Ньютона  newton\_fig = visualize\_optimization(target\_function, newton\_history, "Метод Ньютона")   plt.show()   if \_\_name\_\_ == "\_\_main\_\_":  main()*** |
| *Расчет метрики:* | *Расчет метрики:* |
|  | |