Отчет по решению задачи оптимизации методом сопряженных градиентов

Тишина Ульяна

October 4, 2024

Contents

1 Введение

В данном отчете рассматривается метод сопряженных градиентов для решения задачи оптимизации квадратичной функции. Описаны постановка задачи, реализация метода на языке Python, и приведены результаты тестовых примеров с визуализацией процесса оптимизации.

2 Постановка задачи

Рассмотрим задачу оптимизации функции многих переменных (пусть x - вектор, состоящий из нескольких переменных):

$$f(x) \to min, \tag{1}$$

3 Метод сопряженных градиентов

Метод сопряженных градиентов — это итеративный метод, используемый для нахождения минимума функций.

Этот метод применим только для выпуклых функций, поэтому сначала нужна проверка на выпуклость. Необходимо задать гессиан (матрица 2-х частных проиводных) и проверить ее на положительную определенность. Сделаем это так: возьмем несколько точек и проверим знак гессиана в них. Если хот бы в одной точке гессиан отрицателен, то такая функция не положительно определена. Если такая функция выдает ответ: функция положительно определена, то это не точно, ведь мы могли просто не проверить функцию на тех точках, где она не положительно определена.

4 Реализация проверки на выпуклость

Код на Python, реализующий метод сопряженных градиентов, представлен ниже:

```
x = np.linspace(-10, 10, 10)
y = np.linspace(-10, 10, 10)
z = np.linspace(-10, 10, 10)
x_range = np.array(np.meshgrid(x, y, z)).T.reshape(-1, 3)
y_range = np.array(np.meshgrid(x, y)).T.reshape(-1, 2)

def check_convexity(func, grad_func, hessian_func, x_range):
  for x in x_range:
      H = hessian_func(x)
       eigenvalues = np.linalg.eigvals(H)
       if not np.all(eigenvalues >= 0):
         return False
  return True
```

5 Реализация метода сопряженных граиентов

Код на Python, реализующий метод сопряженных градиентов, представлен ниже:

```
def conjugate_gradient3(f, grad_f, x0, tol=1e-6, max_iter=1000, alpha_max=1.0,
c=0.5, rho=0.5, alpha_min=1e-8):
    arr_f = [f(x0)]
    x = x0
    r = -grad_f(x)
    d = r

    for i in range(max_iter):
        alpha = alpha_max
        while f(x + alpha * d) > f(x) + c * alpha * np.dot(grad_f(x), d) and
        \alpha > alpha_min:
            alpha *= rho

        if alpha < alpha_min:
            break

        x = x + alpha * d
        arr_f.append(f(x))
        r_new = r - alpha * grad_f(x + d)
        beta = np.dot(r_new, r_new) / np.dot(r, r)
        d = r_new + beta * d
        r = r_new

        if np.linalg.norm(r) < tol:
            break

    f_min = f(x)
    return x, f_min, arr_f
```

6 Проверка работы

$$f = x^2 + y^2 + z^2,$$

$$g = x^3 + y^3 - 3xy,$$

$$h = x^4$$

6.1 Пример 1

6.1.1 Реализация

```
def f(x):
  x,y,z = x
  return x**2 + y**2 + z**2

def grad_f(x):
  x,y,z = x
  return np.array([2*x, 2*y, 2*z])

def hessian_f(x):
  x,y,z = x
  return np.array([[2, 0, 0],
                   [0, 2, 0],
                   [0, 0, 2]])
```

6.1.2 Проверка выпуклости

```
x_range = np.array(np.meshgrid(x, y, z)).T.reshape(-1, 3)
is_convex = check_convexity(f, grad_f, hessian_f, x_range)
print(f\"f(x, y, z) vipuklaya: {is_convex}\")
```

Вывод: f(x, y, z) выпуклая: True

Значит, метод к ней предположительно применим. (Помним, что возможно, мы просто не проверили на тех точках, где функция не выпукла).

Начальную точку возьмем (1,1,6)

```
x0_1 = np.array([1, 1, 6])
```

6.1.3 Применение МСГ

```
x_min1, f_min1, arr_f1 = conjugate_gradient3(f, grad_f, x0_1)

print(f"x_min1: {to_float(x_min1)}, f_min1: {to_float(f_min1)}
in {len(arr_f1)} iterations")
```

Выдает:

```
x_min1: ['0.00', '0.00', '0.00'], f_min1: 0.00 in 2 iterations
```

Что соответствует нашим ожиданиям: функция >= 0, поэтому минимум в 0.

6.1.4 График значений функции на каждой итерации

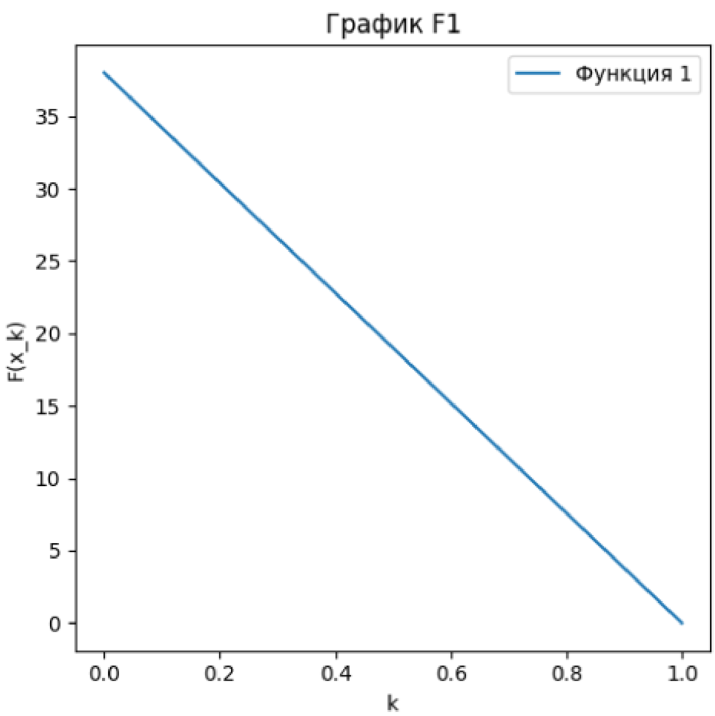

Figure 1: Процесс оптимизации для Примера 1

Минимальное значение 0 было найдено за 1 итерацию, что видно на графике.

6.2 Пример 2

6.2.1 Реализация

```
def g(x):
  x, y = x
  return x**3 + y**3 - 3*x*y

def grad_g(x):
  x, y = x
  return np.array([3*x**2 - 3*y, 3*y**2 - 3*x])

def hessian_g(x):
  x, y = x
  return np.array([[6*x, -3],
                   [-3, 6*y]])
```

6.2.2 Проверка выпуклости

```
y_range = np.array(np.meshgrid(x, y)).T.reshape(-1, 2)
is_convex = check_convexity(g, grad_g, hessian_g, y_range)
print(f"g(x, y) vipuklaya: {is_convex}")
```

g(x, y) выпуклая: False

Значит, метод к ней точно не применим, так как функция не выпуклая. Но мы все равно запустим метод для этой функции, чтобы посмотреть, как метод себя поведет

Начальную точку возьмем (8,-2)

```
x0_2 = np.array([8, -2])
```

6.2.3 Применение МСГ

```
x_min2, f_min2, arr_f2 = conjugate_gradient3(g, grad_g, x0_2)
print(f"x_min2:␣{to_float(x_min2)},␣f_min1:␣{to_float(f_min2)}
in␣{len(arr_f2)}␣iterations")
```

Выдает:

```
x_min2: ['nan', 'nan'], f_min1: nan in 1001 iterations
```

Что соответствует нашим ожиданиям: метод не отработал корректно, значения = nan, а также написано, что за 1001 итерацию, то есть метод остановился не потому, что нашел достаточно близкое значение, а потому, что закончились итерации (в функции задано максимальное кол-во итераций = 1000)

6.2.4 График значений функции на каждой итерации

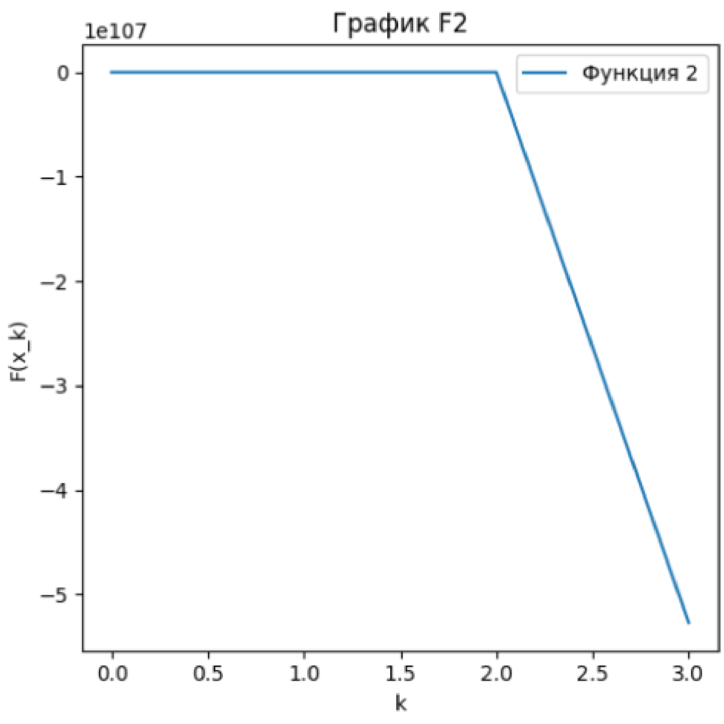

Figure 2: Процесс оптимизации для Примера 2

Первые 3 итерации были какие-то численные значения, а дальше все значения nan, что не отображено на графике. На самом деле минимальное значения кубической функции = -бесконченость, на графике функция как раз уменьшается.

6.3 Пример 3

6.3.1 Реализация

```
def h(x):
  return x**4

def grad_h(x):
  return 4*x**3

def hessian_h(x):
  return np.array([[12*x**2]])
```

6.3.2 Проверка выпуклости

```
x = np.linspace(-10, 10, 10)
is_convex = check_convexity(h, grad_h, hessian_h, x)
print(f"h(x) vipuklaya: {is_convex}")
```

Вывод: h(x) выпуклая: True.

Значит, метод к ней предположительно применим. (Помним, что возможно, мы просто не проверили на тех точках, где функция не выпукла).

Начальную точку возьмем -0.5

```
x0_3 = -0.5
```

6.3.3 Применение МСГ

```
x_min3, f_min3, arr_f3 = conjugate_gradient3(h, grad_h, x0_3)

print(f"x_min3:␣{to_float(x_min3)},␣f_min1:␣{to_float(f_min3)}
in␣{len(arr_f3)}␣iterations")
```

Выдает:

```
x_min3: -0.00, f_min1: 0.00 in 41 iterations
```

Что соответствует нашим ожиданиям: функция $>= 0$, поэтому минимум в 0.

6.3.4 График значений функции на каждой итерации

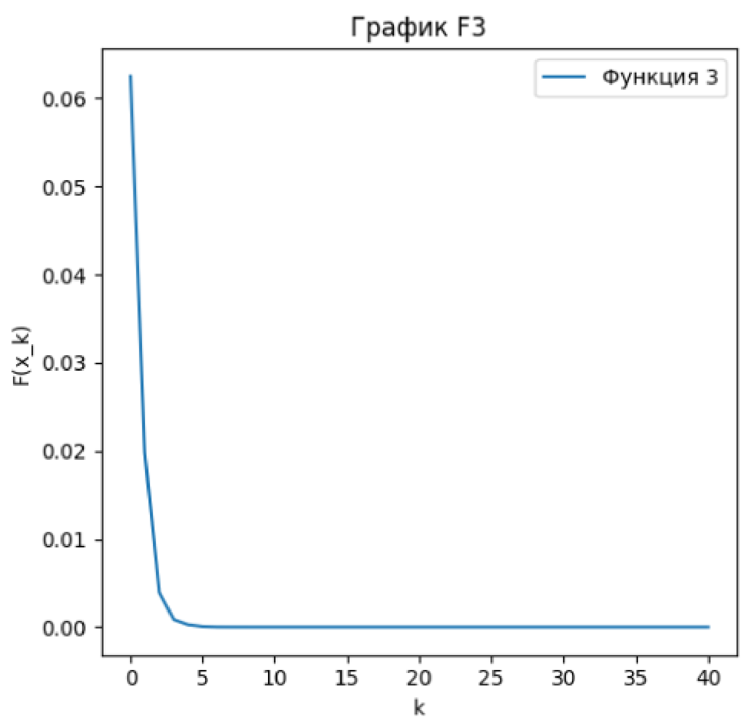

Figure 3: Процесс оптимизации для Примера 3

Минимальное значение 0 было найдено за 40 итераций, что видно на графике.

7 Заключение

В данном отчете была продемонстрирована реализация метода сопряженных градиентов для оптимизации различных функци1. Полученное решение подтверждает эффективность метода только для выпуклых функций. Визуализация показывает процесс сходимости к оптимальному решению, можем сделать вывод, что сходимость линейная.