Отчет по решению задачи оптимизации методом сопряженных градиентов

Тишина Ульяна

November 8, 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)
x_range = np.array(np.meshgrid(x, x, x)).T.reshape(-1, 3)
y_range = np.array(np.meshgrid(x, x)).T.reshape(-1, 2)
check_x_range = np.array(np.meshgrid(x, x, x, x)).T.reshape(-1, 4)

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 функций над векторами и матрицами

```
# mult float and vector
def mult(a, arr):
    return np.array([i*a for i in arr])

# add 2 vectors
def add(arr1, arr2):
    try:
        return np.array([arr1[i]+arr2[i] for i in range(len(arr1))])
    except TypeError:
        arr1+arr2[0]

# dot of 2 vectors
def mydot(arr1, arr2):
    try:
        ans=0
        for i in range(len(arr1)):
            ans+=arr1[i]*arr2[i]
        return ans
    except:
        return arr1*arr2[0]
```

6 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=100, alpha_max=1.0,
                        c=0.5, rho=0.5, alpha_min=1e-8):
        arr_f = [f(x0)]
    x = x0
    r = mult(-1, grad_f(x))
    d = r

    for i in range(max_iter):
        alpha = alpha_max
        while f(add(x, mult(alpha, d))) > f(x) + c * alpha * mydot(grad_f(x), d)
                and alpha > alpha_min:
            alpha *= rho

        if alpha < alpha_min: break

        x = add(x, mult(alpha, d))
        arr_f.append(f(x))
        r_new = add(r, mult(-alpha, grad_f(add(x,d))))
        beta = mydot(r_new, r_new) / mydot(r, r)
        d = add(r_new, mult(beta, d))
        r = r_new

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

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

7 Проверка работы

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

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

$$check\_fun = -2x_0^3 + 4x_0^2 + x_1^4 - 3x_1^2 + 2 * x_2^2 * x_1 - 5x_2 + x_3^4 - x_3^2 * x_0^2 - 10$$

7.1 Пример 1

7.1.1 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]])
```

7.1.2 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])
```

7.1.3 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.

7.1.4 График значений функции на каждой итерации

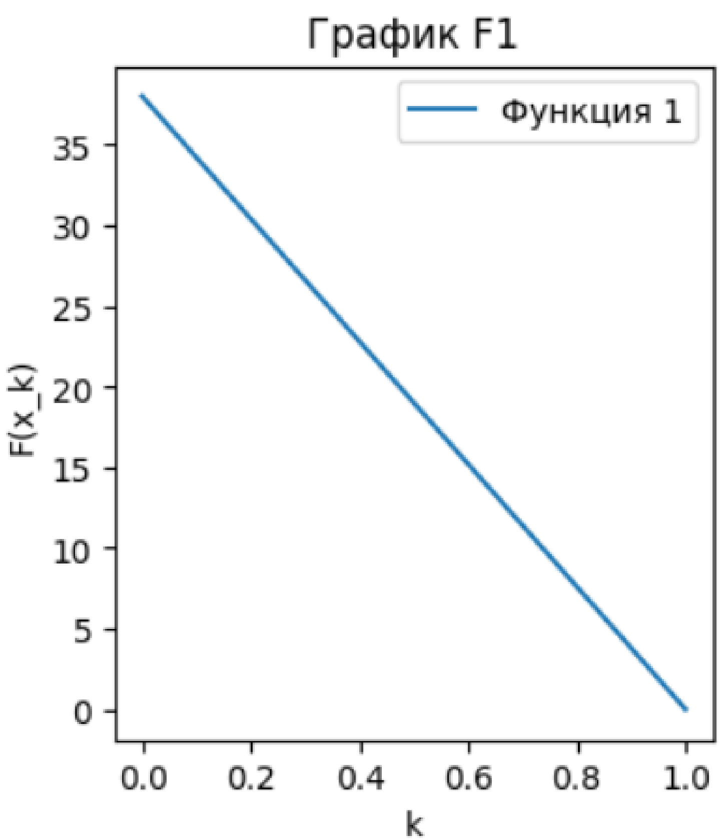

Figure 1: Процесс оптимизации для Примера 1

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

7.2 Пример 2

7.2.1 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]])
```

7.2.2 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])
```

7.2.3 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 101 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 = 100)

7.2.4 График значений функции на каждой итерации

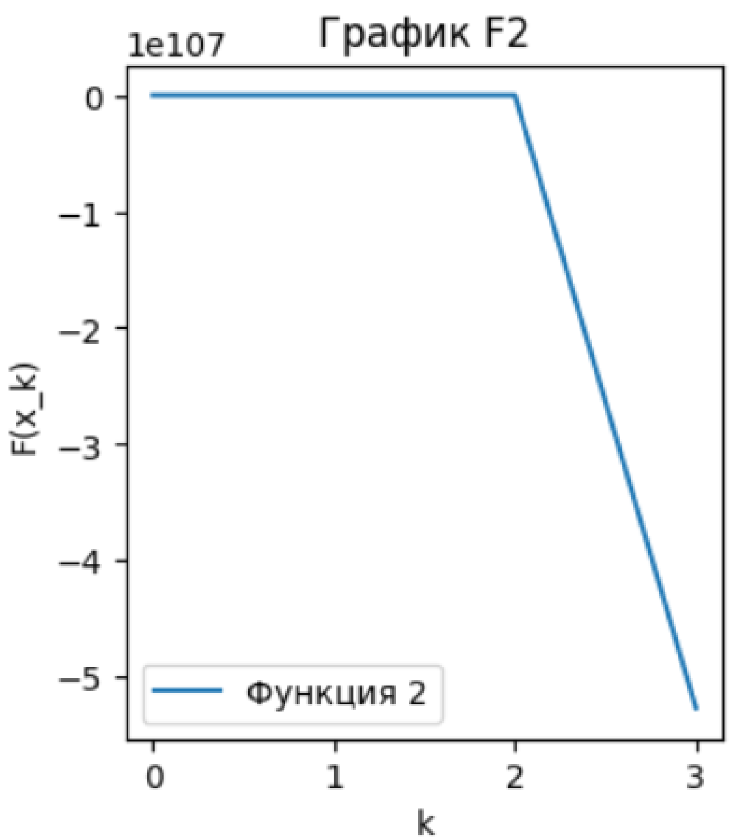

Figure 2: Процесс оптимизации для Примера 2

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.

7.3 Пример 3

7.3.1 Реализация

```
def check_fun(x):
    return -2*x[0]**3 + 4*x[0]**2 + x[1]**4 - 3*x[1]**2 +
          2*x[2]**2 * x[1] - 5*x[2] + x[3]**4 - x[3]**2 * x[0]**2 - 10
def grad_check_fun(x):
    return np.array([-6*x[0]**2 + 8*x[0] - 2*x[3]**2 * x[0],
            4*x[1]**3 - 6*x[1] + 2*x[2]**2,
            4*x[2] * x[1] - 5,
            4*x[3]**3 - 2*x[3] * x[0]**2])
def hessian_check_fun(x):
    return np.array([[-12*x[0] + 8 - 2*x[3]**2, 0, 0, -4*x[3]*x[0]],
            [0, 12*x[1]**2 - 6, 4*x[2], 0],
            [0, 4*x[2], 4*x[1], 0],
            [-4*x[3]*x[0], 0, 0, 12*x[3]**2 - 2*x[0]**2]])
```

7.3.2 Проверка выпуклости

```
check_x_range = np.array(np.meshgrid(x, x, x, x)).T.reshape(-1, 4)
is_convex = check_convexity(check_fun, grad_check_fun, hessian_check_fun, check_
print(f"check_fun(x, y, z) is convex: {is_convex}")
```

Вывод: check_fun(x, y, z) выпуклая: False.
Значит, метод к ней предположительно не применим.
Начальную точку возьмем (-1,-1,-1,-1)

```
check_x_0 = [-1,-1,-1,-1]
```

7.3.3 Применение МСГ

```
check_x_min, check_f_min, check_arr_f = conjugate_gradient3(check_fun,
 grad_check_fun, check_x_0)

print(f"check_x_min:␣{to_float(check_x_min)},␣check_f_min:
␣␣␣␣␣␣␣␣{to_float(check_f_min)}␣in␣{len(check_arr_f)}␣iterations")
```

Выдает:

```
check_x_min: ['nan', 'nan', 'nan', 'nan'], check_f_min: nan in 101 iterations
```

Что соответствует нашим ожиданиям: функция приняла значение nan, то есть бесконечность, поэтому минимума нет (точнее, он = -бесконечность)

7.3.4 График значений функции на каждой итерации

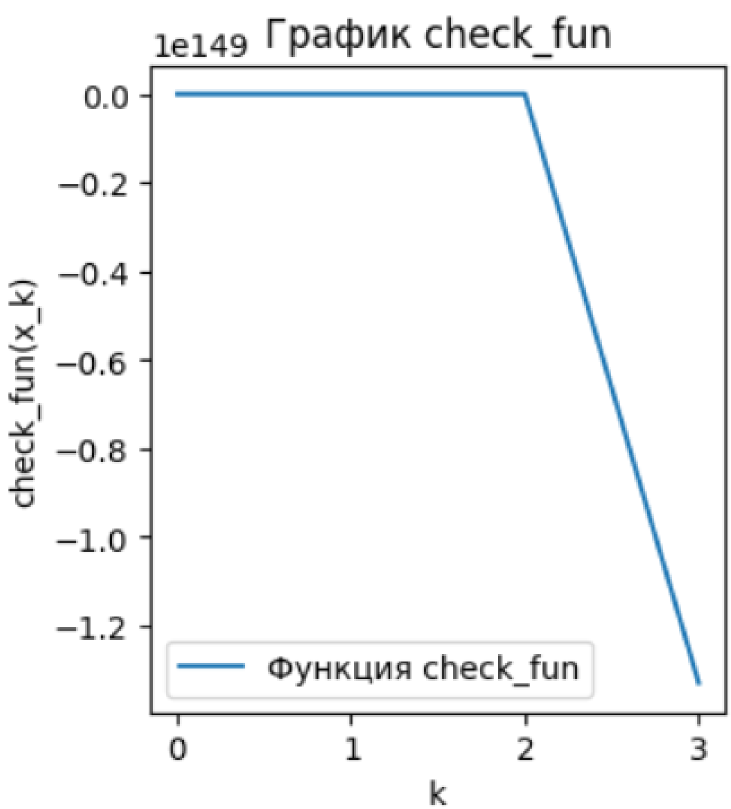

Figure 3: Процесс оптимизации для Примера 3

Минимальное значение не было найдено, что видно на графике (после 3-й итерации график обрывается, потому что значения nan не отображаются)

8 Заключение

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