Отчет по решению задачи оптимизации методом сопряженных градиентов

Тишина Ульяна

December 12, 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 хорошо применим 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 хотя 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]
# norm of vector
def norm(arr):
    return sum(i**2 for i in arr)**0.5

# take grad
def my_grad(f, x, h=1e-5):
    grad = np.zeros_like(x)
    for i in range(len(x)):
        xh = x.copy()
        x_h = x.copy()
        xh[i] += h
        x_h[i] -= h
        grad[i] = (f(xh) - f(x_h))/h/2
    return grad
```

6 Реализация метода сопряженных граиентов

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

```
def conjugate_gradient5(f, grad_f, x0, tol=1e-6, max_iter=200,
        alpha_max=1.0, c=0.5, rho=0.5, alpha_min=1e-8, max_f = 1e15):

        arr_f = [f(x0)]
    x = x0
    r = mult(-1, my_grad(f, x))
    d = r

    ans = {"message": "ok",
           "status": 0,
           "nit": 0,
           "x": x,
           "f_min": arr_f[-1],
           "arr_f": arr_f}

    for i in range(max_iter):
        ans["nit"]+=1
        alpha = alpha_max

        while f(add(x, mult(alpha, d))) > f(x) + c * alpha *
                        mydot(my_grad(f, x), d) and alpha > alpha_min:
            alpha *= rho

        if abs(f(x))>=max_f:
            ans["message"] = 'too big f or too small'
            ans["x"], ans["f_min"], ans["arr_f"], ans["status"] = x, arr_f[-1],
            return ans
        if alpha < alpha_min:
            ans["message"] = 'too small alpha'
            ans["x"], ans["f_min"], ans["arr_f"], ans["status"] = x, arr_f[-1],
            return ans

        x = add(x, mult(alpha, d))
        arr_f.append(f(x))
        r_new = mult(-1, my_grad(f, x))
        beta = mydot(r_new, r_new) / mydot(r, r)
        d = add(r_new, mult(beta, d))
        r = r_new

        if norm(r) < tol:
```

```
        break

ans["x"], ans["f_min"], ans["arr_f"] = x, arr_f[-1], arr_f
return ans
```

7 Проверка работы

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

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

$$fun3 = -2x_0^3 + 4x_0^2 + x_1^4 - 3x_1^2 + 2 * x_2^2 * x_1 - 5x_2 + x_3^4 - x_3^2 * x_0^2 - 10$$

$$fun4 = (x_0 + 3)^4 + (x_1 - 1)^2 + (x_2 - 2)^2 + (x_3 + x_0)^2$$

$$fun5 = -2x_0^3 + 4x_0^2 + x_1^4 - 3x_1^3 - 4x_2^3 + 2x_2^4 - 5x_3 + x_3^2 + 10 + x_0 + x_1 + x_2 + x_3$$

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

Вывод: f(x, y, z) is convex: True

Начальную точку возьмем (1,1,6)

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

7.1.3 Применение МСГ

```
res1 = conjugate_gradient5(f, grad_f, x0_1)
```

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

g(x, y) is convex: False

Начальную точку возьмем (8,-2)

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

7.2.3 Применение МСГ

```
res2 = conjugate_gradient5(g, grad_g, x0_2)
```

7.3 Пример 3

7.3.1 Реализация

```
def check_fun(x):
    x0,x1,x2,x3=x
    return -2*x0**3 + 4*x0**2 + x1**4 - 3*x1**2 + 2*x2**2 * x1 -
               5*x2 + x3**4 - x3**2 * x0**2 - 10
def grad_check_fun(x):
    x0,x1,x2,x3=x
    return np.array([-6*x0**2 + 8*x0 - 2*x3**2 * x0,
            4*x1**3 - 6*x1 + 2*x2**2,
            4*x2 * x1 - 5,
            4*x3**3 - 2*x3 * x0**2])
def hessian_check_fun(x):
    x0,x1,x2,x3=x
    return np.array([[-12*x0 + 8 - 2*x3**2, 0, 0, -4*x3*x0],
            [0, 12*x1**2 - 6, 4*x2, 0],
            [0, 4*x2, 4*x1, 0],
            [-4*x3*x0, 0, 0, 12*x3**2 - 2*x0**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, k) is convex: {is_convex}")
```

Вывод: check_fun(x, y, z, k) выпуклая: False.

Начальную точку возьмем (-1,-1,-1,-1)

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

7.3.3 Применение МСГ

```
check_res = conjugate_gradient5(check_fun, grad_check_fun, check_x_0)
```

7.4 Пример 4

7.4.1 Реализация

```
def fun4(x):
    x0, x1, x2, x3 = x
    return (x0+3)**4 + (x1-1)**2 + (x2-2)**2 + (x3+x0)**2
def grad_fun4(x):
    x0, x1, x2, x3 = x
    return [4*(x0+3)**3 + 2*(x3+x0),
            2*(x1-1), 2*(x2-2), 2*(x3+x0)]
def hessian_fun4(x):
    x0, x1, x2, x3 = x
    return [[12*(x0+3)**2 + 2,0,0,2],
            [0,2,0,0], [0,0,2,0], [2,0,0,2]]
```

7.4.2 Проверка выпуклости

```
fun4_x_range = np.array(np.meshgrid(x, x, x, x)).T.reshape(-1, 4)
is_convex = check_convexity(fun4, grad_fun4, hessian_fun4, fun4_x_range)
print(f"fun4(x, y, z, k) is convex: {is_convex}")
```

Вывод: fun4(x, y, z, k) выпуклая: True

Начальную точку возьмем (-1,0.5,0.5,0.5)

```
fun4_x0 = [-1,0.5,0.5,0.5]
```

7.4.3 Применение МСГ

```
res4 = conjugate_gradient5(fun4, grad_fun4, fun4_x0)
```

7.5 Пример 5

7.5.1 Реализация

```
def fun5(x):
    x0,x1,x2,x3=x
    return
     -2*x0**3+4*x0**2+x1**4-3*x1**3-4*x2**3+2*x2**4-5*x3+x3**2+10+x0+x1+x2+x3
def grad_fun5(x):
    x0,x1,x2,x3=x
    return [-6*x0**2+8*x0+1,  4*x1**3-9*x1**2+1,
            -12*x2**2+8*x2**3+1,      -5+2*x3+1]
def hessian_fun5(x):
    x0,x1,x2,x3=x
    return [[-12*x0+8,0,0,0],    [0,12*x1**2-18*x1,0,0],
            [0,0,-24*x2+24*x2**2,0],   [0,0,0,2]]
```

7.5.2 Проверка выпуклости

```
fun5_x_range = np.array(np.meshgrid(x, x, x, x)).T.reshape(-1, 4)
is_convex = check_convexity(fun5, grad_fun5, hessian_fun5, fun5_x_range)
print(f"fun5(x, y, z, k) is convex: {is_convex}")
```

Вывод: fun5(x, y, z, k) is convex: False

Начальную точку возьмем (-0.5,2.5,1.5,2.0)

```
fun5_x0=[-0.5,2.5,1.5,2.0]
```

7.5.3 Применение МСГ

```
res5 = conjugate_gradient5(fun5, grad_fun5, fun5_x0)
```

8 Результаты работы программы на примерах

8.1 Результаты

```
______________________________
         resses for fun1
______________________________
message : ok
status : 0
nit : 14
x : [-3.67238076e-08 -3.67238076e-08 -1.87282572e-07]
f_min : 3.777203792212465e-14
arr_f : [38, 0.75, 0.1524390243898475, 0.010501657291865944, 0.00106218
______________________________
         resses for fun2
______________________________
message : too big f or too small
status : 1
nit : 3
x : [-58194131.7081309    3242624.75525305]
f_min : -1.970436465556398e+23
arr_f : [552, -6745437.0, -1.970436465556398e+23]
______________________________
         resses for fun3
______________________________
message : too big f or too small
status : 1
nit : 3
x : [ 4.23473057e+04 -1.01930764e+04 -1.20000000e+01  7.18505095e+03]
f_min : -7.92705188228059e+16
arr_f : [-3, -2103.0, -7.92705188228059e+16]
______________________________
         resses for fun4
______________________________
message : ok
status : 0
nit : 129
x : [-2.99472138  1.00000011  2.00000033  2.9947211 ]
f_min : 7.765895255094651e-10
arr_f : [18.75, 8.561424314820293, 5.013255506297735, 2.237968559061228
______________________________
         resses for fun5
______________________________
message : ok
status : 0
nit : 21
x : [-0.11506934  2.19826555  1.43969263  2.        ]
f_min : -2.281788806256098
arr_f : [-0.4375, -1.9115416371215739, -2.1783969318084497, -2.25231665
```

Figure 1: Процесс оптимизации для примеров

8.2 Графики

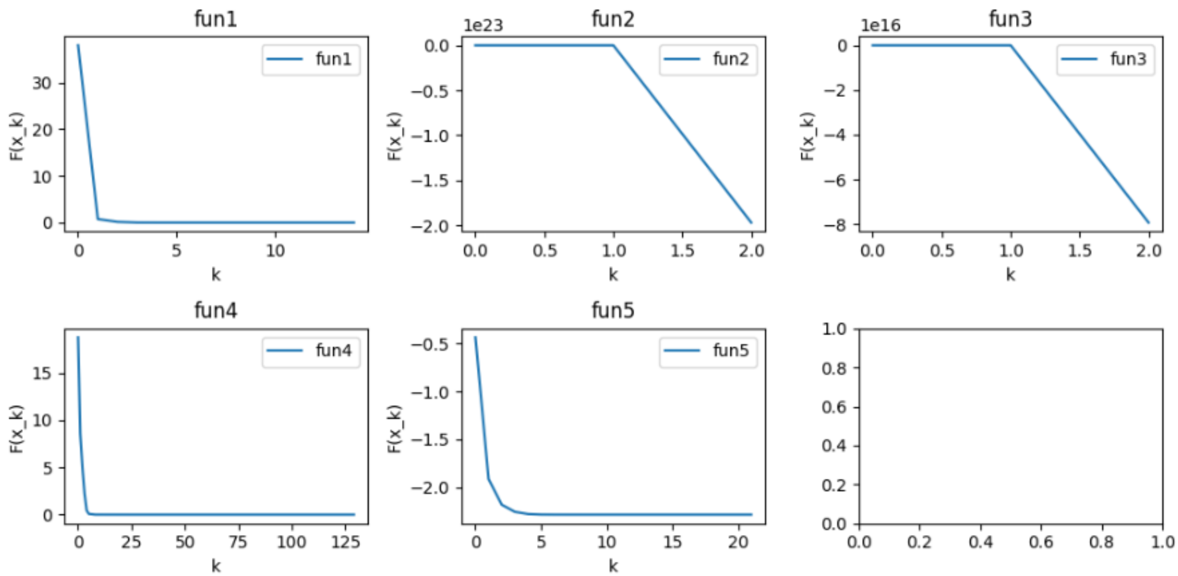

Figure 2: Процесс оптимизации для примеров

9 Эксперименты с параметрами на примере 5

Возьмем начальные точки от -50 до 50 с шагом 0,2

Table 1: Вывод только успешных результатов

x_0	x_{min}	f_{min}	nit	status	mes	f_{right}	$stat_{right}$
[-1, -1, -1, -1]	[-0.12, -0.31, -0.27, 2.00]	5.55	93	0	ok	-2.94e+202	2
[-0.4, -0.4, -0.4, -0.4]	[-0.12, -0.31, -0.27, 2.00]	5.55	38	0	ok	5.55	0
[-0.2, -0.2, -0.2, -0.2]	[-0.12, -0.31, -0.27, 2.00]	5.55	30	0	ok	5.55	0
[0, 0, 0, 0]	[-0.12, -0.31, -0.27, 2.00]	5.55	38	0	ok	5.55	0
[0.2, 0.2, 0.2, 0.2]	[-0.12, -0.31, -0.27, 2.00]	5.55	37	0	ok	5.55	0
[0.6, 0.6, 0.6, 0.6]	[-0.12, 2.20, 1.44, 2.00]	-2.28	155	0	ok	-2.28	0
[1.0, 1.0, 1.0, 1.0]	[-0.12, 2.20, 1.44, 2.00]	-2.28	157	0	ok	-2.28	0
[1.2, 1.2, 1.2, 1.2]	[-0.12, 2.20, 1.44, 2.00]	-2.28	91	0	ok	-2.28	0

С бОльшей точностью см. на гитхабе, файл task3.ipynb

Теперь поэкспериментируем с шагом rho, на который мы изменяем alpha. Точность будет задана 0,0001. Начальную точку возьмем (0,0,0,0), т.к. в ней (см. выше) метод сходится.

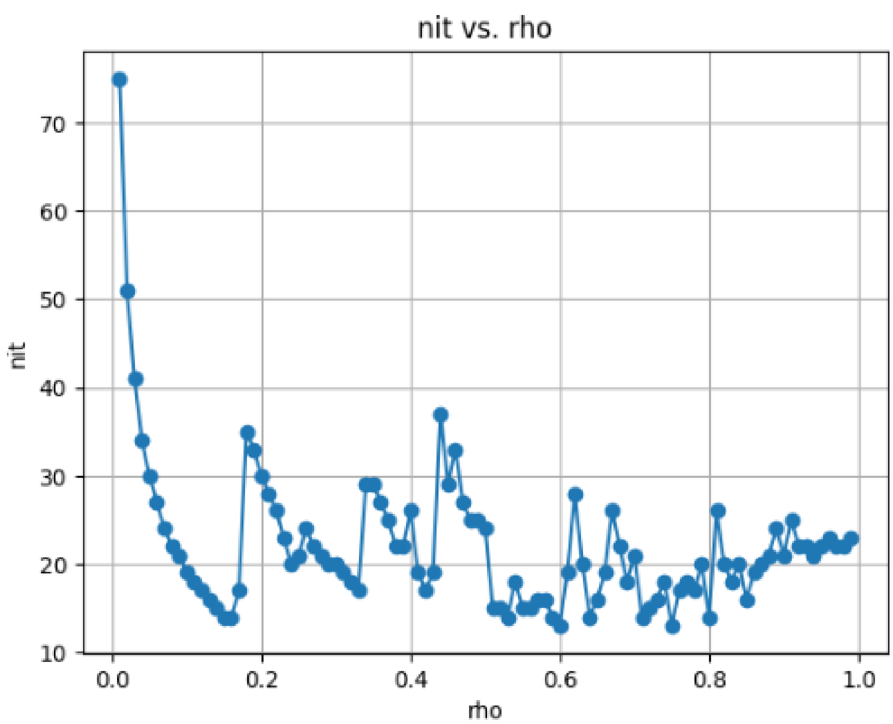

Figure 3: Вывод только успешных результатов

Теперь попробуем несколько видов остановки метода и посмотрим на точность выводимых результатов.

Возьмем норму градиента, норму шага и разность значений f.

Начальную точку возьмем = (0, 0, 0, 0).

Значение f, которое выдает встроенный сетод minimize, равен 5.548828333952681.

Вывод реализованного метода:

```
________________________________
norm_grad
________________________________
message ok
status 0
nit 22
x [−0.11506929  −0.31235628  −0.26604444   1.99999997]
f_min 5.54882833393948
x0 [0.0, 0.0, 0.0, 0.0]
________________________________
norm_step
________________________________
message ok
status 0
nit 8
x [−0.11536704  −0.31233456  −0.26557075   1.99641519]
f_min 5.548842508550054
x0 [0.0, 0.0, 0.0, 0.0]
________________________________
dif_f
________________________________
message ok
status 0
nit 12
x [−0.11507592  −0.31236762  −0.26602964   1.99987639]
f_min 5.548828350747011
x0 [0.0, 0.0, 0.0, 0.0]
```

Точность была задана = 1e-7. Хуже всех результат при выборе остановки нормы шага. Выбор разности f и нормы градиента дают нужную точность, но при сравнении с эталонным результатом - лучше оказалась норма градиента.

10 Выводы

Пример 1: выпуклая функция, минимум найден успешно

Пример 2,3: минимум не был найден - решение ушло на - бесконечность

Пример 4: минимум найден успешно

Пример 5: если взять точку, далекую от решения, то решение не будет найдено - функция завершится из-за не найденного alpha. Если подобрать точку, близкую к минимуму, то программа успешно находит минимум, как показано на картинках.

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 сверхлинейная.

На примере 5 эмпирическим путем было показано, что лучше брать начальную точку, близкую к решению (хотя ее можно и подобрать, просто потребуется несколько запусков), rho брать $= 0.6$, а метод сстановки брать $=$ норма градиента. При выборе таких параметров метод будет наимболее эффективен.