Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования Московский
государственный технический университет имени Н.Э. Баумана

Лабораторная работа №5. Вариант 6.
«Методы поиска условного экстремума»
по курсу
«Методы оптимизации»

Студент группы ИУ9-82 Иванов Г.М

Преподаватель Каганов Ю.Т.

Москва, 2019

Contents

1 Цель работы

1. Изучение алгоритмов условной оптимизации.
2. Разработка программ реализации алгоритмов условной оптимизации.
3. Нахождение оптимальных условий решений для задач с учетом ограничений.

2 Постановка задачи

Дано: 1 Вариант. Функция Розенброка на множестве R^2:

$$f(x) = \sum_{i=1}^{n-1} [a(x_i^2 - x_{i+1})^2 + b(x_i - 1)^2] + f_0, \tag{1}$$

где

$$a = 30, \quad b = 2, \quad f_0 = 80, \quad n = 2, \tag{2}$$

тогда функция $f(x)$ будет выглядеть следующим образом:

$$f(x) = 30 * (x_0^2 - x_1)^2 + 2 * (x_0 - 1)^2 + 80 \tag{3}$$

Функции ограничений:

$$\begin{cases} g_1(x_1, x_2) = x_1^2 + x_2^2 - 5 \leq 0 \\ g_2(x_1, x_2) = -x_1 \leq 0 \\ g_3(x_1, x_2) = -x_2 \leq 0 \end{cases} \tag{4}$$

1. Найти условный экстремум методами:
   (a) Штрафных функций;
   (b) Барьерных функций;
   (c) Модифицированных функций Лагранжа;
   (d) Проекции градиента.
2. Найти все стационарные точни и значения функций, соотвестсвующие этим точкам.
3. Оценить скорость сходимости указанных алгоритмов.
4. Реализовать алгоритмы с помощью языка программирования высокого уровня.

3 Исследование

Найдем глобальные экстремумы функции

$$f(x) = 30 * (x_0^2 - x_1)^2 + 2 * (x_0 - 1)^2 + 80 \tag{5}$$

с помощью сервиса WolframAlpha.com:

$$min(f(x)) = 80, \quad (x_0, x_1) = (1, 1) \tag{6}$$

3.1 Метод щтрафных функций.

Идея метода заключается в сведении задачи на условный минимум к решению последовательности задач поиска безусловного минимума вспомогательной функции:

$$F(x, r^k) = f(x) + P(x, r^k) \to \min_{x \in R^n}, \tag{7}$$

где $P(x, r^k)$ - штрафная функция, r^k - параметр штрафа, задаваемый на каждой k- й итерации. Это связано с возможностью применения эффективных и надежных методов поиска безусловного экстремума,

3.2 Метод барьерных функций.

Идея метода заключается в сведении задачи на условный минимум к решению последовательности задач поиска безусловного минимума вспомогательной функции:

$$F(x, r^k) = f(x) + P(x, r^k) \to \min_{x \in R^n}, \tag{8}$$

где $P(x, r^k)$ - штрафная функция, r^k - параметр штрафа. Используется обратная штрафная функция $P(x, r^k) = -r^k \sum_{j=1}^{m} \frac{1}{g_i(x)}$.

3.3 Метод модифицированных функций Лагранжа.

Стратегия аналогична используемой в методе внешних штрафов, только штрафная функция добавляется не к целевой функции, а к классической функции Лагранжа. В результате задача на условный минимум сводится к решению последовательности задач поиска безусловного минимума модифицированной функции Лагранжа:

$$L(x, \lambda^k, \mu^k, r^k) = f(x) + \sum_{j=1}^{l} \lambda_j^k g_j(x) + \frac{r^k}{2} \sum_{j=1}^{l} g_j^2(x) + \frac{1}{2r^k} \sum_{j=l+1}^{m} (max^2(0, \mu_j^k + r^k g_j(x)) - \mu^2) \quad (9)$$

где λ^k - векторы множителей Лагранжа; r^l - параметр штрафа; k - номер итерации.

3.4 Метод проекции градиента.

Стратегия поиска решения задачи учитывает тот факт, что решение x^* может лежать как внутри, так и на границе множества допустимых решений. Для определения приближенного решения x^* строится последовательность точек

$$\{x^*\} : x^{k+1} = x^k + \delta x^k, \quad k = 1, .., \quad (10)$$

где приращение δx^k определяется в каждой точке x^k в зависимости от того, где ведется поиск – внутри или на границе множества допустимых решений.

Решение задачи начинается с обхода границы допустимой области. Обход границ множества допустимых решений связан с выявлением активных в точке ограичений, аппроксимацией их плоскостью

$$A_k \delta x = \tau_k \quad (11)$$

Поиск ограничений, активных в точке, рассматривается как самостоятельная задача, которая может быть решена путем последовательных приближений. Процедура вычисления точек последовательности обеспечивает последовательность движения вдоль границы допустимой области. При выполнении неравенства $|| \triangle x^k|| = || - [E - A_k^T(A_k A_k^T)^{-1} A_k] \bigtriangledown f(x^k)$, где $\epsilon 2$ заданное достаточно малое положительное число, вычисляется приближение λ^k вектора множителей Лагранжа λ^*:

$$\lambda k = -(A_k A_k^T)^{-1} A_k \bigtriangledown f(x^k). \tag{12}$$

Если $\lambda k \leq 0$, то в точке x^k выполнены необходимые условия минимума и в ней должны быть проверены достаточные условия. Если среди множителей λ_j^k есть отрицательные, то это означает, что x^k не является приближением точки x^*, так как в ней не выполняются условия минимума $f(x)$ при ограничениях $g_j(x) \leq 0$,j = 1..m. Однако выбор шага αk позволяет говорить о том, что значение f(x) не может быть уменьшено при заданном составе активных ограничений и, следовательно, процесс минимизации f(x) необходимо продолжить, уменьшив их число: в число пассивных ограничений переводят то из ограничений, которому соответствует наибольший по абсолютной величине отрицательный множитель λ_j^k. Такая процедура поиска позволяет отыскать решение, лежащее как на границе, так и внутри множества допустимых решений.

4 Практическая реализация

Все методы были реализованы на языке программирования **Python**.

Листинг 1. Метод штрафных функций.

```
def method_penalty_function(x_c, r, c, eps, f, h_1, h_2, h_3):
    print_function.print_start("Penalty function")
    k = 1
    while True:
        p = lambda x: r / 2 * (h_1(x) ** 2 + h_2(x) ** 2 + h_3(x) ** 2)
        x_new = nelder_mead(lambda x: f(x) + p(x_c), x_c)
        penalty_value = p(x_new)
        if abs(penalty_value) < eps:
            print_function.print_end_function(k, x_new, f)
            return x_new
        else:
            r *= c
            x_c = x_new
            k += 1
```

Листинг 2. Метод барьерных функкий.

```
def method_barrier_function(x_c, r, c, eps, f, h_1, h_2, h_3):
    print_function.print_start("Barrier function")
    k = 1
    while True:
        if (h_1(x_c) ** 2 + h_2(x_c) ** 2 + h_3(x_c) ** 2) != 0:
            p = lambda x, r_: -(r_ ** k) * (1 / (h_1(x) ** 2 + h_2(x) ** 2 +
            ↪ h_3(x) ** 2))
        else:
            p = lambda x, r_: -(r_ ** k)
        x_new = nelder_mead(lambda x: f(x) + p(x_c, r), x_c)
        penalty_value = p(x_new, r)
        if abs(penalty_value) < eps:
            print_function.print_end_function(k, x_new, f)
            return x_new
        else:
            k += 1
            x_c = x_new
            r /= c
```

Листинг 3. Метод модифицированных функций Лагранжа.

```python
def lagrange_functions(x_start, increase_param, r, lambdas, mu, eps, f, h_1, h_2, h_3):
    eps /= 100
    print_function.print_start("Lagrange Functions")
    k = 1
    x_c = x_start
    while True:

        def func(x):
            return f(x) + langrange_lambda + eq_penalty(x) + 1 / (2 * r) * sum(np.array(neq_penalty) - np.array(mu_squared))

        langrange_lambda = np.sum(np.matmul(np.array(lambdas), [h_1(x_start), h_2(x_start), h_3(x_start)]))
        eq_penalty = lambda x: (r / 2) * (h_1(x) ** 2 + h_2(x) ** 2 + h_3(x))
        neq_penalty = [max(0.0, mu[0] + r * pow(h_1(x_c), 2)),
                       max(0.0, mu[1] + r * pow(h_2(x_c), 2)),
                       max(0.0, mu[2] + r * pow(h_3(x_c), 2))]

        mu_squared = [mu[0] ** 2, mu[1] ** 2, mu[2] ** 2]

        x_new = nelder_mead(func, x_c)

        new_penalty = [max(0.0, mu[0] + r * pow(h_1(x_new), 2)),
                       max(0.0, mu[1] + r * pow(h_2(x_new), 2)),
                       max(0.0, mu[2] + r * pow(h_3(x_new), 2))]

        new_mu_squared = [mu[0] ** 2, mu[1] ** 2, mu[2] ** 2]

        penalty_value = r / 2 * (h_1(x_new) ** 2 + h_2(x_new) ** 2 + h_3(x_new)) * \
                        sum(np.array(new_penalty) - np.array(new_mu_squared))

        if abs(penalty_value) < eps:
            print_function.print_end_function(k, x_new, f)
            return x_new
        else:
            k += 1
            x_c = x_new
            r *= increase_param
            lambdas += np.array(np.multiply(r, [h_1(x_start), h_2(x_start), h_3(x_start)]))
```

Листинг 4. Метод проекции градиента.

```python
def gradientProjections(x0, eps1, eps2, f, h_1, h_2, h_3, grad, cond1, cond2, cond3):
    print_function.print_start("Gradient Projections")
    max_iterations = 1000
    k = 0
    t_k_star = 3.4
    while k < max_iterations:
        if k >= max_iterations:
            print_function.print_end_function(k, x0, f)
            return x0, k
        a_k = np.array([cond1(x0), cond2(x0), cond3(x0)])
        t_k = np.multiply(-1, np.array([h_1(x0), h_2(x0), h_3(x0)])).T
        delta_2_x_k = np.matmul(np.matmul(a_k.T, np.linalg.inv(np.matmul(a_k, a_k.T))), t_k)
        norm_value = np.linalg.norm(delta_2_x_k)
        gradient_value = grad(x0)
        # if gradient_value == 0:
        #     print_function.print_end_function(k, x0, f)
        #     return x0
        V = a_k.T @ np.linalg.inv(a_k @ a_k.T)
        X = V @ a_k
        S = np.eye(X.shape[0]) - X
        delta_x_k = (-1 * S) @ gradient_value
        # if 0 > delta_x_k > eps1:
        #     active_restrictions.remove(gradient_value)
        if np.linalg.norm(delta_x_k) <= eps1 and norm_value <= eps2:
            print_function.print_end_function(k, x0, f)
            return x0, k
        elif np.linalg.norm(delta_x_k) > eps1 and norm_value <= eps2:
            delta_2_x_k = 0
            new_func = lambda t: f(x0 + t * delta_x_k)
            interval = method_svann(1.0, 0.001, new_func)
            t_k_star = golden_section_method(0.001, interval, new_func)
        elif np.linalg.norm(delta_x_k) > eps1 and norm_value > eps2:
            new_func = lambda t: f(x0 + t * delta_x_k)
            interval = method_svann(1.0, 0.001, new_func)
            t_k_star = golden_section_method(0.001, interval, new_func)
        elif np.linalg.norm(delta_x_k) <= eps1 and norm_value > eps2:
            delta_x_k = 0
        x0 = x0 + t_k_star * delta_x_k + delta_2_x_k
        k += 1
```

5 Результаты.

При последовательном запуске всех алгоритмов со следующими параметрами:

$$\epsilon = 10^{-4} \tag{13}$$

были получены следующие результаты:

Листинг 5. Результаты выполнения программ.

```
Start Penalty function
         Iteration(s): 54
         f([0.99993326 0.99983504]) = {80.00000003866163}

Start Barrier function
         Iteration(s): 1
         f([0.99993326 0.99983504]) = {80.00000003866163}

Start Lagrange Functions
         Iteration(s): 20
         f([1.02641507 1.05415417]) = {80.00140727823911}

Start Gradient Projections
         Iteration(s): 33
         f([1.00634279, 0.99907874]) = {80.00566773836404}
```

Все результаты с небольшой погрешностью совпадают с результатами полученными с помощью сервиса WolframAlpha.com в пункте 3.