Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
кафедра Исследования операций

Метод штрафных функций с правилом скорейшего спуска

Выполнил:
студент 411 группы
Ефремова Ольга Игоревна

Москва
2022

1 Математическое описание метода

1.1 Метод штрафных функций

Метод штрафных функций является одним из наиболее простих и широко используемых методов задач минимизации. Основная идея мелода заключается в сведении исходной задачи

$$F(x) \to inf\ ;\ x \in X$$

$$X = \{x \in X_0 : g_i(x) \leq 0, i = \overline{1,n}; h_i(x) = 0, i = \overline{1,m}\}$$

к последоватнльности задач минимизации

$$F_k(x) \to inf\ ;\ x \in X_0\ ;\ k = 0, 1, \ldots,$$

где $F_k(x)$ – некоторая вспомогательная функция, а множество X_0 содержит X. При этом функция $F_k(x)$ подобрана так, чтобы с ростом k мало отличалась от исходной функции $F(x)$ на множестве X и быстро возрастала на $X_0 \backslash X$.

Последовательность функций $P_k(x), k = 0, 1, \ldots$, определенных и неотрицательных на множестве X_0 называют штрафной функцией множества X на множестве X_0, если

$$\lim_{k\to\infty} P_k(x) = \begin{cases} 0, & x \in X \\ \infty, & x \in X_0 \backslash X \end{cases}$$

Будем использовать вспомогательную функцию вида $\mathrm{F}_k(x) = F(x) + A_k * P(x) \to inf,$
где $P_k(x) = A_k * P(x)$:
$A_k \to +\infty\ k \to \infty$ и $A_k > 0$ для любого k;
$P(x) = \sum_{i=1}^{n}(max(g_i(x), 0)^p) + \sum_{i=1}^{m}(|h_i(x)|^p),\ p \geq 1$.

Сходимость метода для данной штрафной функции опиана в [1]

1.2 Правило скорейшего спуска

Пусть задача оптимизации имеет вид:

$$F(x) \to inf; x \in X,$$

Основная идея метода заключается в том, чтобы идти в направлении наискорейшего спуска, а это направление задаётся антиградиентом $-\nabla F$:

$$x_{k+1} = x_k - \alpha_k \nabla F(x),$$

где α_k задает скорость градиентного спуска.

Для поиска минимума воспользуемся функцией $F_k(\alpha)$, минимизируя её:

$$F_k(\alpha) = F(x_k - \alpha \nabla F(x_k)) \to inf$$

От сюда вычисляется $\alpha_k = argmin F_k(\alpha)$, необходимое для очередногт шага спска.

2 Описание программного кода

2.1 Скорейший спуск (minimize)

```
def mininize(f,eps, x_old):
  leng = (len(x_old))
  grad = nd.Gradient(f)
  x_new = x_old
  k = 0
  while k < 100:
    fk = lambda lam: f(x_old - lam*grad(x_old))
    res = sc.minimize_scalar(fk)
    l = res.x
    x_new = x_old - l*grad(x_old)
    if np.linalg.norm(x_new - x_old) < eps:
      break
    k = k + 1
    x_old = x_new
  return x_new
```

На вход функция получает lambda-функцию $\boldsymbol{f}$, которую необходимо минимизировать, значение $\boldsymbol{eps}$ и начальную точку $\boldsymbol{x_{old}}$.

Далее вычисляется длина вектора решений, градиент функции. С помощью цикла на не более чем 100 итераций (чтобы избежать зацикливания) происходит рассчет следующего приближения по схеме описанной в разделе **1.2**. Параллельно проверяем точность полученного на $\boldsymbol{k}-$ итерации значения (расстояное между двумя последовательными приближениями).

После выхода из цикла возвращаем приближение, полученное на последнем шаге цикла.

2.2 Метод штрафных функций (penalty method)

```
def penalty_method(f, g, h, eps, x_0):
    A = lambda k: k*k*30
    P = lambda x: sum(max(gi(x), 0)**p for gi in g) + sum(abs(hi(x))**p for hi in h)
    k = 1
    p=2
    x_old = x_0
    data = pd.DataFrame({’точка’: [ ’---------------------------’, x_old], ’значение
        функции’: [’------------------’, ’%.4f’ %f(x_old)]})
    while(True):
        Fk = lambda x: f(x) + A(k)*P(x)
        x_new = mininize(Fk, eps, x_old)
        new_row = {’точка’: np.round(x_new, 4), ’значение функции’:f(x_new)}
        data = data.append(new_row, ignore_index=True)
        if A(k)*P(x_new) < eps:
            break
        x_old = x_new
        k = k + 1
    print(data)
    print(’\Точкаn минимума: ’, x_new)
    print(’Значение функции: ’, f(x_new))
    print(’Число итераций: ’, k)
```

На вход функция получает lambda-функцию $\boldsymbol{f}$, которую необходимо минимизировать, ограничения в виде массивов лямбда-функций $\boldsymbol{g}$ и $\boldsymbol{h}$ значение $\boldsymbol{eps}$ и начальную точку $\boldsymbol{x_{old}}$.

Далее вычисляются lamda-функции $\boldsymbol{A(k)}$ и $\boldsymbol{P(x)}$, необходимые для формирования штрафной функции, описанной в разделе **1.1**. Здесь функция $\boldsymbol{A(k)}$ является настраиваемой и подбирается методом проб. В цикле без условия останова на каждой итерации вычисляется вспомогательная функция $\boldsymbol{F_k(x)}$, описанная в разделе **1.1**, после чего с помощью функции,реализующей скорейший спуск вычисляется новое значение приближения. После чего вычисляется точность полученного приближения. Если она удовлетворяет заданной точности ($\boldsymbol{A_k * P(x) < eps}$), происходит выход из цикла. Иначе – переход на следующий шаг.

Все полученные приближения записываются в таблицу по ходу вычислений. В конце выводится таблица полученных приближений, конечное приближение (точка минимума), значение исходной функции в этой точке и число проведенных итераций в методе штрафных функций.

2.3 Подстановка задачи в программу

```
f  = lambda x: 150*(sum((x[i] - (i+1)*x[0])**4 for i in range(1,6))) + (x[0] - 2)**2
g1 = lambda x: sum((x[i])**2 for i in range(0,6)) - 363
g = [g1]
h = []

eps = 0.001
x_0 = np.array((1.0, 2.0, 3.0, 4.0, 5.0, 6.0))
x_1 = np.array((2.0, 4.0, 6.0, 8.0, 10.0, 11.0))
penalty_method(f, g, h, eps, x_0)
penalty_method(f, g, h, eps, x_1)
```

Все функции задаются в виде lambda-функций. $\boldsymbol{f}$ – целевая функция задачи, $\boldsymbol{g_i}$ - функции, отвечающие ограничениям-неравенствам, $\boldsymbol{h_i}$ - функции, отвечающие ограничениям-равенствам. Все ограничения записываются в соответствующие массивы, устанавливается целевая точность, задается начальная точка. После чего все параметры передаются в функцию, реализующую метод штрафных функций.

3 Описание решения задачи оптимизации с помощью написанного на I этапе Практикума программного кода.

Для решения поставленной задачи целевая функция и ограничения были переписаны в программе в необходимом для выполнения кода виде.

Так как метод штрафных функция не способен искать глобальный минимум, а находит лишь локальный, необходимо самостоятельно "прикинуть"расположение минимума. В качестве первой "начальной точки"возьмем точку $\mathbf{(1.0, 2.0, 3.0, 4.0, 5.0, 6.0)}$ чтобы подтвердить предположение, что метод ходится в точках локального минимума. Самостоятельно можно вычислить точку глобального минимума $\mathbf{(2.0, 4.0, 6.0, 8.0, 10.0, 12.0)}$, используем в качестве второй "начальной точки"точку $\mathbf{(2.0, 4.0, 6.0, 8.0, 10.0, 11.0)}$ близкую к глобальному минимуму.

```
                                           точка    значение функции
0                      -------------------------  ------------------
1                   [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]            1.0000
2  [1.2811, 2.536, 3.8173, 5.0998, 6.3829, 7.6665]          0.517049

Точка минимума:  [1.28112003 2.53596576 3.81731896 5.09978133 6.38291793 7.66651491]
Значение функции:  0.5170494057099344
Число итераций:  1
Nice
                                           точка    значение функции
0                      -------------------------  ------------------
1                   [2.0, 4.0, 6.0, 8.0, 10.0, 11.0]          150.0000
2  [1.9605, 3.9066, 5.8651, 7.8239, 9.7831, 11.7423]          0.001643

Точка минимума:  [ 1.96046241  3.90658429  5.8650514   7.82392738  9.78305085 11.74234429]
Значение функции:  0.0016425008973756316
Число итераций:  1
```

После выполнения проходов из обеих точек мы видим, что программа пришла в точки локального минимума (с заданной точностью). Погрешность и уход из глобального минимума на малую величину обусловлен погрешностью машинных вычислений.

4 3 этап: Анализ полученных минимумов на наборе начальных точек

Решаемая задача:
$\boldsymbol{F = 150 \sum_{i=2}^{6} (x_i - ix_1)^4 + (x_1 - 2)^2 \to inf}$ Ограничения:
$\boldsymbol{g_1 = \sum_{i=1}^{6} x_i^2 - 363 \leq 0}$
Начальные точи:
$\boldsymbol{(x, x, x, x, x, x)}$, где $\boldsymbol{x \in (-50, 50)}$

Полученные точки в прикрепленном "result.csv"файле. Видно, что при нахождении вне допустимого множества программа доходит до ближайшего локального минимума внутри допустимого множества. К глобальному минимуму (с заданной точностью) получилось прийти только при $\boldsymbol{x = 8}$. Различные результаты из различных точек говорят о том, что данный метод оптимизации не рассчитан на поиск глобальных минимумов и идет в сторону ближайшего локального минимума.

Список используемой литературы

[1] Ф.П. Васильев - "Численные методы решения экстремальных задач"