КІБЕРНЕТИКА та КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 519.85 DOI:10.34229/2707-451X.23.2.5

А.А. СУПРУН

ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ ДЛЯ $r(\alpha)$-АЛГОРИТМУ З ПРИСКОРЕНОЮ РЕАЛІЗАЦІЄЮ РОЗТЯГУ ПРОСТОРУ

Розглядається модифікація r(α)-алгоритму з прискореною операцією розтягу простору, в якій оператор розтягу простору діє на підпросторі змінних, що визначається найбільшими за абсолютним значенням компонентами вектора різниці двох послідовних субградієнтів. Описується схема алгоритму. Наводяться результати обчислювальних експериментів задач мінімізації квадратичних та кусково-лінійних опуклих функцій на основі яких показано переваги модифікованого варіанта r(α)-алгоритму.

Ключові слова: *оператор розтягу простору, r(α)-алгоритм, кусково-лінійна функція.*

© А.А. Супрун, 2023

Вступ. *r*-алгоритми Н.З. Шора [1, 2] це ефективні методи для оптимізації гладких та негладких погано обумовлених функцій. Вони грунтуються на двох ідеях – процедурі спуску за напрямком антисубградієнта опуклої функції у перетвореному просторі змінних та використанні спеціального оператора розтягу простору в напрямку різниці двох послідовних субградієнтів для перетворення простору змінних.

Один з найбільш поширених варіантів *r*-алгоритмів це $r(\alpha)$-алгоритм з адаптивною процедурою спуску за напрямком [3, 4]. В ньому оператор розтягу простору $R_\alpha(\eta)$ використовує сталий коефіцієнт розтягу α в напрямку вектора η – нормованого вектора різниці двох послідовних субградієнтів у перетвореному просторі змінних.

В статті розглядається модифікація $r(\alpha)$-алгоритму з адаптивною процедурою спуску за прямком, в якому оператор стиску простору $R_\beta(\eta)$, – діє на підпросторі змінних, що визначається найбільшими за абсолютним значенням компонентами вектора η [5].

В розділі 1 описується $r(\alpha)$-алгоритм з адаптивною процедурою спуску за напрямком. В розділі 2 описується схема модифікованого $r(\alpha)$-алгоритму з прискореною процедурою розтягу простору. В розділі 3 наводяться результати обчислювальних експериментів із застосування обох варіантів $r(\alpha)$-алгоритму до задач мінімізації негладких кусково-лінійних та гладких квадратичних функцій. Для проведення експериментів використовуються програмні реалізації алгоритмів мовою програмування C++.

1. $r(\alpha)$-алгоритм. *r*-алгоритми – це сімейство субградієнтних методів з перетворенням простору змінних, в якому використовується спуск по антисубградієнту, а перетворення простору на кожній ітерації реалізується за допомогою оператора розтягу простору в напрямку різниці двох послідовних субградієнтів. Загальну схему *r*-алгоритмів можна описати наступним чином.

Нехай $f(x)$ – опукла функція, $x \in R^n$, x_0 – стартова точка, B_0 – деяка задана $n \times n$-матриця, α_k ($\alpha_k > 1$) – набір коефіцієнтів розтягу простору, k = 0, 1, 2, ...Тоді *r*-алгоритм – це ітеративний процес пошуку мінімуму $f(x)$, що генерує наступну послідовність:

$$x_{k+1} := x_k - h_k B_k \xi_k, \quad B_{k+1} = B_k R_{\beta_k}(\eta_k), \quad k = 0, 1, 2, \dots, \tag{1}$$

де

$$\xi_k := \frac{B_k^T g(x_k)}{\|B_k^T g(x_k)\|}, \quad h_k \approx h_k^* := argmin_{h \geq 0} f(x_k - h_k B_k \xi_k), \tag{2}$$

$$R_\beta(\eta_k) = I + (\beta - 1)\eta_k \eta_k{}^T, \ \eta_k := \frac{B_k^T r_k}{\|B_k^T r_k\|}, \quad r_k := g_{k+1} - g_k. \tag{3}$$

Тут h_k – величина кроку з точки x_k до точки (наближеного) мінімуму у напрямку $-h_k B_k \xi_k$, $R_\beta(\eta) = I + (\beta - 1)\eta\eta^T$ – оператор стиснення простору у напрямку нормованого вектора η з коефіцієнтом стиснення $\beta = 1/\alpha < 1$, g_{k+1} та g_k – субградієнти функції f в точках x_{k+1} та x_k відповідно.

В залежності від вибору коефіцієнта розтягу простору α_k, процедури пошуку (наближеного) мінімуму функції за напрямком в (2) та умов зупинки алгоритму можна отримати різні варіанти *r*-алгоритма.

Одним із поширених варантів *r*-алгоритмів є $r(\alpha)$-алгоритм з адаптивним спуском за напрямком. В ньому, як видно з назви, для розтягу простору використовується постійне значення коефіцієнта розтягу простору $\alpha_k = \alpha$.

Як умови зупинки використовуються стандартні критерії:

- $\|x_{k+1} - x_k\| <$ *epsx*(*epsx* > 0) – умова зупинки за аргументом;
- $\|g_{k+1}\| <$ *epsg*(*epsg* > 0) – умова зупинки за нормою градієнта (для гладких функцій);
- *k* > *maxitn* (*maxitn N*) – умова зупинки за максимальної кількості ітерацій.

Також використовується умова зупинки за максимальною кількістю ітерацій у процедурі пошуку наближеного мінімуму за напрямком. За замовчуванням, цей параметр дорівнює 500.

Як процедура пошуку наближеного мінімуму за напрямком у $r(\alpha)$-алгоритмі використовується спеціальна процедура адаптивного спуску в напрямку антисубградієнта в перетвореному просторі змінних. Адаптивний спуск за напрямком – це ітеративна процедура одновимірного спуску за напрямком, яка завершується, як тільки виконується задана умова. Умовою завершення процедури спуску для $r(\alpha)$-алгоритма є умова, що субградієнт функції у знайденій точці має утворювати із вектором спуску гострий кут. На кожному кроці ітерації процедури відбувається рух вздовж заданого напрямку на величину hs, яка налаштовується адаптивно за допомогою чотирьох параметрів: h_0 – початкова величина крокового множника (використовується на першій ітерації *r*-алгоритму, а на кожній наступній – уточнюється), q_1 ($q_1 \leq 1$) – коефіцієнт зменшення крокового множника (якщо умова завершення спуску за напрямком виконується за перший крок), q_2 ($q_2 \geq 1$) – коефіцієнт збільшення крокового множника. Через кожні *nh* (*nh* > 1) кроків одновимірного спуску величина крокового множника hs збільшується в q_2 раз.

Алгоритм адаптивного спуску за напрямком можна описати так.

Вхідні дані:

p – напрямок спуску,

y_0 – початкова точка.

Параметри алгоритму:

h_0 – початкова величина крокового множника,

q_1 – коефіцієнт зменшення крокового множника (якщо умова завершення спуску за напрямком виконується за перший крок),

q_2 – коефіцієнт збільшення крокового множника,

nh – кількість кроків після яких збільшується кроковий множник.

Вихідні дані:

y_{ls} – точка, в якій виконується умова зупинки процедури спуску.

Початок алгоритму:

$hs = h_0$ – ініціалізація початкового значення *hs*,

$ls = 0$ – ініціалізація початкового значення лічильника циклу.

Початок циклу:

$ls = ls + 1$ – номер ітерації,

$y_{ls} = y_{ls-1} + hs * p$ – перерахунок нової точки.

Якщо y_{ls} задовольняє умову зупинки ⇒ зупинка та вихід із циклу.

Якщо *ls* кратне *nh* ⇒ $hs = hs * q_2$ – оновити значення *hs*.

Кінець циклу.

Якщо ls = 1 => $h_0 = hs * q_1$ – оновити початкове значення кроку *hs* для наступного застосування процедури.

Кінець алгоритму.

2. Опис прискореного варіанта *r*(α)-алгоритму. На кожній ітерації k $r(\alpha)$-алгоритму використовується матриця B_k, яка перераховується за наступною формулою:

$$B_{k+1} = B_k R_\beta(\eta_k) = B_k(I + (\beta - 1)\eta_k \eta_k^T), \tag{4}$$

де $\eta_k = \frac{B_k^T r_k}{\|B_k^T r_k\|}$ – нормалізований вектор різниці двох послідовних субградієнтів в перетвореному просторі, а матриця $R_\beta(\eta_k)$ задає оператор розтягу простору в напрямку η_k.

Для перерахунку B_{k+1} в (4) виконується $2n^2 + n$ операцій множення – з них n^2 для обчислення вектора $B_k^T r_k$; n для обчислення вектора $(\beta - 1)\eta_k$; n^2 для обчислення матриці $\eta_k {\eta_k}^T$. Для нормування вектора η_k необхідно виконати $2n$ операцій множення. Тому загальна кількість операцій множення необхідних для обчислення B_{k+1} дорівнює $2n^2 + 3n$.

В роботі [5] відмічено, що кількість операцій множення в (4) можна зменшити, якщо у векторі η_k залишити лише *m (m < n)* найбільших за модулем елементів. Наприклад, вектор η_k можна модифікувати за допомогою спеціального параметра контролю t наступним чином:

$$\tilde{\eta}_k = \{\tilde{\eta}_k^i\}_{i=1}^n, \text{ де } \tilde{\eta}_k^i = \begin{cases} 0, & |\eta_k^i| \le t\, max_{\mathrm{j}=1,\mathrm{n}} |\eta_k^j| \\ \eta_k^i, & |\eta_k^i| \ge t\, max_{\mathrm{j}=1,\mathrm{n}} |\eta_k^j| \end{cases} \quad 0 \le t < 1, \quad \eta_k = \{\eta_k^i\}_{i=1}^n. \quad i = 0,\ldots, n. \tag{5}$$

Таким чином, на перерахунок коефіцієнтів матриці B_{k+1} в (4) будуть впливати тільки *m* ненульових елементів вектора $\tilde{\eta}_k$ і, як наслідок, кількість необхідних операцій множення, зменшиться з $2n^2 + 3n$ до $2nm + 2m + n$. Зазначимо, що в такому випадку, на кожній ітерації алгоритму, оператор розтягу простору $R_\beta(\tilde{\eta}_k)$ діє не на всьому просторі змінних, як у загальній схемі *r*-алгоритму, а тільки на деякому *m*-вимірному підпросторі.

На основі вищенаведеного, опишемо схему $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком та прискореним варіантом розтягу простору у напрямку наближеної різниці двох послідовних субградієнтів:

Вхідні параметри:

x_0 – стартова точка,

calcfg(*x*) – функція, шо обчислює значення та субградієнт цільової функції *f*(*x*) у точці *x*.

Параметри для операції розтягу простору:

α – коефіцієнт розтягу простору,

t – коефіцієнт контролю напрямку розтягу простору.

Параметри процедури адаптивного спуску за напрямком:

h_0 – початкове значення крокового множника,

q_1 – коефіцієнт зменшення крокового множника,

q_2 – коефіцієнт збільшення крокового множника,

nh – період збільшення крокового множника.

Параметри критеріїв зупинки алгоритма:

maxitn – максимальна кількість ітерацій основного циклу,

epsg – параметр критерію зупинки за нормою градієнта (працює для гладких функцій),

epsx – параметр критерію зупинки за аргументами.

Вихідні параметри:

fr – рекордне значення функції,

xr – рекордна точка.

Початок алгоритму:

Ініціалізуємо локальні змінні алгоритму:

$k = 0$ – лічильник ітерацій основного циклу,

$B_0 = I$ – матрицю B_0 ініціалізуємо як одиничну діагональну,

$hs = h_0$ – кроковий множника для адаптивної процедури спуску за напрямком,

$w = {}^{1}/_{\alpha} - 1$ – множник оператора стиску простору.

Обчислюємо значення функції *f* та її субградієнта g_1 в стартовій точці x_0:

$f,\ g_1 = calcfg\ (x_0)$.

Ініціалізуємо дані про рекордну точку:

$fr = f,\ xr = x$.

Перевіряэмо критерій зупинки за epsg:

Якщо $\|g_1\| < epsg \Rightarrow$ ЗАВЕРШИТИ АЛГОРИТМ

Початок головного циклу:

Перевіряємо критерій зупинки за кількістю ітерацій у циклі:

Якщо $k > maxitn \Rightarrow$ ЗАВЕРШИТИ АЛГОРИТМ

Оновлюємо лічильник *k*:

$k = k + 1$.

Обчислюємо субградієнт g_1 у перетвореному просторі змінних:

$g = B_k^T g_1$.

Нормуємо вектор g:

$g = \frac{g}{\|g\|}$.

Обчислюємо відповідний вектор у початковому просторі змінних:

$dx = B_k^T g$.

Ініціалізуємо лічильник *ls* та початкову точку x_{k+1}:

$ls = 0,\ x_{k+1} = x_k$.

Початок циклу процедури спуску за напрямком $-dx$:

Перераховуємо x_{k+1} та *ls*.

$x_{k+1} = x_{k+1} - hs\ dx,$

$ls = ls + 1$.

Обчислюємо значення функції $f(x)$ та її субградієнта g у точці x_{k+1}:

$f,\ g_2 = calcfg\ (x_{k+1})$.

Оновлюємо дані про рекордну точку.

Якщо $f < fr \Rightarrow fr = f, xr = x_{k+1}$.

Перевіряємо умову зупинки за нормою градієнта (для гладких функцій):
Якщо $\|g_2\| <$ *epsg* ⇒ ЗАВЕРШИТИ АЛГОРИТМ
Перевіряємо умову зупинки за необмеженістю *f*:
Якщо *ls* > 500 ⇒ ЗАВЕРШИТИ АЛГОРИТМ
Оновлюємо кроковий множник *hs*:
Якщо *ls* кратне *nh* ⇒ $hs = hs * q_2$.
Обчислюємо скалярний добуток напрямку спуску dx та субградієнта g_2:
$d = {g_2}^T dx$.
Перевіряємо умову завершення спуску за напрямком:
Якщо d < 0 ⇒ завершити цикл.
Переходимо до наступної ітерації циклу.
Кінець циклу процедури спуску за напрямком $-dx$.
Оновлюємо кроковий множник *hs* для процедури спуску на наступної ітерації:
Якщо *ls* = 1 ⇒ $hs = hs * q_1$.
Перевіряємо умову зупинки за аргументом:
Якщо $\|x_{k+1} - x_k\| <$ *epsx* ⇒ ЗАВЕРШИТИ АЛГОРИТМ.
Обчислюємо вектор різниці субградієнтів:
$r_k = g_2 - g_1$.
Обчислюємо r_k у перетвореному просторі змінних:
$\eta = B_k^T r_k$.
Нормуємо вектор η:
$\eta_k = \frac{\eta}{\|\eta\|}$.
Модифікуємо вектор r згідно з (5):

$$\tilde{\eta}_k = \{\tilde{\eta}_k^i\}_{i=1}^n, \text{ де } \tilde{\eta}_k^i = \begin{cases} 0, & |\eta_k^i| \le t\, max_{\text{j = 1,n}}|\eta_k^j| \\ \eta_k^i, & |\eta_k^i| \ge t\, max_{\text{j = 1,n}}|\eta_k^j| \end{cases} \quad \eta_k = \{\eta_k^i\}_{i=1}^n. \quad i = 0, ..., n.$$

Обчислюємо матрицю B_{k+1}:
$B_{k+1} = B_k R_\beta(\tilde{\eta}_k) = B_k(I + w\, \tilde{\eta}_k \tilde{\eta}_k^T)$.
Зберігаємо поточний субградієнт:
$g_1 = g_2$.
Переходимо до наступної ітерації циклу.
Кінець головного циклу
Кінець алгоритму

3. Обчислювальні експерименти. В даному розділі наведемо результати обчислювальних експериментів задач мінімізації двох сімейств опуклих яружних функцій – кусково-лінійних негладких функцій SABS(*q*,*n*) = $\sum_{i=1}^n q^{i-1} abs(x_i - 1)$ та квадратичних строго опуклих функцій SQUAD(*q*,*n*) = = $\sum_{i=1}^n q^{2(i-1)} (x'_i - 1)^2$ з параметром q = 1.1 при розмірностях n = 100 та n = 200 та точкою мінімуму x_0 = (1,…,1). Для їх мінімізації використовувались C++ реалізація $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком ralgb5a та реалізація модифікованого $r(\alpha)$-алгоритму з адаптивною процедурою спуску за напрямком та прискореним розтягом простору ralgb5at. Абревіатура "b5" означає, що коректується $n \times n$-матриця B_k, а обчислювальна складність кожної ітерації k визначається $5n^2$ арифметичними операціями множення (з них, як зазначалось, $2n^2$ припадає на перерахунок матриці B_k в (4)). “a” означає, що пошук наближеного мінімуму за напрямком реалізується адативною процедурою пошуку за напрямком. “t” означає, що реалізується операція прискореного розтягу простору, напрям якого контролюється параметром t за формулою (5). Обчислення прово-

дились на комп'ютері з процесором Intel(R) Core(TM) i7-8565U CPU 1.80GHz 1.99GHz та оперативною пам'яттю 8 Гб з використанням компілятора g++ (GCC) 10.1.0. При компіляції вихідного коду використовувалась команда оптимізації –03.

Параметри алгоритмів вибирались згідно з рекомендаціями в [3] – α = 2, h_0 = 10 для n = 100 та h_0 = 15 для n = 200, q_1 = 1 для негладких функцій SABS та q_1 = 0.85 для гладких функцій SQUAD, q_2 = 1.1, nh = 3, *epsx* = 10^{-6}, *epsg* = 10^{-12}, maxitn = 15000. Стартова точка вибиралась x_0 = (0,…,0).

В табл. 1–4 наведно результати тестів мінімізації 4 вищенаведених функцій для 6 варіантів *r(α)*-алгоритму при різних значеннях параметра контроля напрямку розтягу простору t = {0, 0.5, 02, 01, 0.02, 0.01}. При t = 0 використовується класичний ralgb5a, при інших значеннях t – ralgb5at. Результати описуються наступними показниками: t – значення параметру контроля напрямку розтягу простору, itn – кількість виконаних алгоритмом ітерацій, ncalls – кількість викликів функції calcfg, що обчислює значення мінімізуючої функції та її субградієнта в точці, nzeros – сумарна кількість нульових компонент вектора розтягу простору за всі ітерації, totalcomp – сумарна кількість операцій множення з ненульовими елементами за всі ітерації для перерахунку B_k в (4), economcomp – відсоткове відношення значення totalcomp для даного алгоритма відносно відповідного значення для ralgb5a, time – час роботи алгоритма в секундах.

ТАБЛИЦЯ 1. Розрахунки для SABS(1.1, 100) = $\sum_{i=1}^{100} 1.1^{i-1} abs(x_i - 1)$

t	itn	ncalls	nzeros	totalcomp	economcomp	time, s
0(ralgb5a)	2778	2 785	0	56 393 400	100	0.37
0.5	2826	2 827	201 293	16 686 314	29.58	0.378
0.2	2 772	2 778	106 444	34 749 612	61.61	0.372
0.1	2 774	2 782	61 540	43 860 820	77.77	0.365
0.02	2 784	2 791	18 456	52 766 788	93.59	0.366
0.01	2 750	2 755	11 595	53 462 510	94.8	0.355

ТАБЛИЦЯ 2. Розрахунки для SABS(1.1, 200) = $\sum_{i=1}^{200} 1.1^{i-1} abs(x_i - 1)$

t	itn	ncalls	nzeros	totalcomp	economcomp	time, s
0(ralgb5)	6 953	6967	0	560 411 800	100	1.113
0.5	7040	7042	1 083 135	132 003 930	24	1.034
0.2	6 947	6952	625 929	308 304 942	55	1.04
0.1	6 124	6923	398577	397 204 246	70.87	1.02
0.02	6932	6944	172 708	489 290 784	87.3	1.024
0.01	6 938	6929	133 835	504 677 730	90.05	1.016

ТАБЛИЦЯ 3. Розрахунки для SQUAD(1.1, 100) = $\sum_{i=1}^{100} 1.1^{2(i-1)} (x'_i - 1)^2$

t	itn	ncalls	nzeros	totalcomp	economcomp	time, s
0(ralgb5)	528	1 032	0	10 718 400	100	0.027
0.5	310	563	28 833	468 834	4.37	0.014
0.2	313	560	26 586	983 628	9.17	0.014
0.1	335	613	26 401	1 467 598	13.69	0.014
0.02	494	921	26 041	4 768 018	44.48	0.021
0.01	539	1 006	20 998	6 700 204	62.51	0.023

ТАБЛИЦЯ 4. Розрахунки для SQUAD(1.1, 200) = $\sum_{i=1}^{200} 1.1^{2(i-1)} (x'_i - 1)^2$

t	itn	ncalls	nzeros	totalcomp	economcomp	time, s
0(ralgb5)	2 286	4 792	0	184 251 600	100	0.384
0.5	695	1326	133 730	2 257 740	1.222 324	0.115
0.2	722	1386	131 864	5 184 072	2.806619	0.119
0.1	950	1 925	161 331	11715 138	6.34249	0.156
0.02	1 644	3 407	205 935	49 720 730	26.91844	0.259
0.01	2 233	4 615	186 602	104 965 996	56.82783	0.354

Зазначимо, що всі ітерації алгоритмів зійшлись до точки мінімуму функцій і завершилися за умовою зупинки за аргументом. З таблиць видно, що для алгоритмів з модифікованим варіантом розтягу простору ralgb5at (рядки при $t > 0$) сумарна кількість операцій множення (стовпчик totalcomp) значно менша у порівнянні з класичним ralgb5a (рядок при $t = 0$). Слід зазначити, що для негладких функцій SABS загальна кількість ітерацій (стовпчик itn) і кількість обчислень значень функції та субградієнта (стовпчик ncalls) приблизно однакова, а для квадратичних функцій SQUAD загальна кількість ітерацій, кількість обчислень значень функції та градієнта, і, як наслідок, час виконання (стовпчик time) варіантів ralgb5at значно менший у порівнянні з класичним ralgb5a. Це може свідчити про те, що для таких класів функцій прискорена реалізація розтягу простору краще враховує характер яружності функцій.

Висновок. В статті розглянуто модифікацію $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за напрямком спуску. В $r(\alpha)$-алгоритмі розтяг простору здійснюється у напрямку нормованого вектора різниці двох послідовних субградієнтів. В модифікованому $r(\alpha)$-алгоритмі розтяг простору здійснюється у напрямку, який визначається найбільшими за модулем компонентами вектора різниці двох послідовних субградієнтів. Кількість таких компонент на кожній ітерації модифікованого $r(\alpha)$-алгоритму змінюється і контролюється за рахунок спеціального параметра. Таким чином, розтяг простору на кожній ітерації відбувається у відповідному підпросторі змінних меншої розмірності, а кількість операцій множення для процедури розтягу простору на кожній ітерації зменшується з $2n^2 + 3n$ до $2nm + 2m + n$, де n – розмірність простору, m – кількість ненульових компонент вектора розтягу простору.

Наведено схему модифікованого $r(\alpha)$-алгоритму та результати обчислювальних експериментів для задач мінімізації опуклих кусково-лінійних та квадратичних яружних функцій. Можна зазначити, що у порівнянні зі звичайним $r(\alpha)$-алгоритмом, його модифікований варіант для мінімізації кусково-лінійної функції загалом потребує приблизно таку ж саму кількість ітерацій, але, при цьому, виконує значно меншу кількість операцій множення. Для мінімізації квадратичної строго опуклої функції кількість ітерацій і, як наслідок, час виконання програми, зменшуються. Тому алгоритми на основі прискореного розтягу простору в напрямках наближених до напрямку нормованої різниці двох послідовних субградієнтів можна вважати доволі перспективним. Такі алгоритми можуть краще враховувати геометрію (характер) яружності функції та при відповідній програмній реалізації мати менший час виконання.

На основі прискореного розтягу простору можна також розробляти гібридні субградієнтні алгоритми найшвидшого спуску, в яких перетворення простору відбувається лише на деякому підпросторі змінних. Такі алгоритми потенційно можуть бути доволі ефективними, якщо підпростір, в якому функція є яружною має відносно невелику розмірність. Наразі проводяться дослідження таких модифікацій *r*-алгоритму для мінімізації яружних опуклих функцій.

Подяка. Автор висловлює подяку співробітникам Інституту кібернетики імені В.М. Глушкова НАН України Стецюку П.І. за наукові консультації щодо матеріалу статті та Стовбі В.О. за допомогу у оформленні рукопису статті.

Список літератури

1. Шор Н.З. Методы минимизации недифференцируемых функций и их приложения. Киев: Наукова думка, 1979. 200 с.
2. Shor N.Z. Non-Differentiable Optimization and Polynomial Problems. Kluwer Academic Publishers, Boston, Dordrecht, London. 1998. 412 p.
3. Стецюк П.И. Теория и программные реализации *r*-алгоритмов Шора. *Кибернетика и системный анализ*. 2017. № 5. С. 43–57.
4. Стецюк П.І., Пилиповський О.В., Хом'як О.М. GNU Octave та Python реалізації r-алгоритму Шора з адаптивним регулюванням кроку. *Cybernetics and Computer Technologies*. 2022. 3. С. 98–112. https://doi.org/10.34229/2707-451X.22.3.10
5. Стецюк П.І., Жмуд О.О. Про прискорену реалізацію оператору розтягу простору. Питання прикладної математики і математичного моделювання. Тези доповідей XVIII міжнародної науково-практичної конференції «*Математичне та програмне забезпечення інтелектуальних систем* (*MPZIS-2020*)», м. Дніпро, 18–20 листопада 2020 р. Дніпро: ДНУ, 2020.

Одержано 18.07.2023

Супрун Антон Андрійович,
аспірант Інституту кібернетики імені В.М. Глушкова НАН України, Київ.
anton_s2007@ukr.net

УДК 519.85

А.А. Супрун

Обчислювальні експерименти для *r*(α)-алгоритму з прискореною реалізацією розтягу простору

Інститут кібернетики імені В.М. Глушкова НАН України, Київ
Листування: anton_s2007@ukr.net

В статті розглянуто модифікацію *r*(α)-алгоритму з адаптивним регулюванням кроку за напрямком спуску, яка використовує прискорену реалізацію розтягу простору. В класичних варіантах *r*(α)-алгоритмів розтяг простору здійснюється у напрямку нормованого вектора різниці двох послідовних

субградієнтів. У модифікованому $r(\alpha)$-алгоритмі розтяг простору здійснюється у напрямку, який визначається найбільшими за модулем компонентами вектора різниці двох послідовних субградієнтів. Кількість таких компонент на кожній ітерації модифікованого $r(\alpha)$-алгоритму змінюється і контролюється за рахунок спеціального параметра. Таким чином, розтяг простору на кожній ітерації відбувається у відповідному підпросторі змінних меншої розмірності, а кількість операцій множення для процедури розтягу простору на кожній ітерації зменшується з $2n^2 + 3n$ до $2nm + 2m + n$, де n – розмірність простору, m – кількість ненульових компонент вектора розтягу простору.

Наведено схему модифікованого $r(\alpha)$-алгоритму та результати обчислювальних експериментів для задач мінімізації опуклих кусково-лінійних та квадратичних яружних функцій. Для розв'язку задач використовувались C++ реалізації класичного $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за напрямком спуску ralgb5a та його модифікація з прискореною реалізацію розтягу простору ralgb5at. На основі отриманих результатів зазначено, що у порівнянні зі звичайним $r(\alpha)$-алгоритмом, його модифікований варіант для мінімізації кусково-лінійної функції загалом потребує приблизно таку ж саму кількість ітерацій, але, при цьому, виконує значно меншу кількість операцій множення. Для мінімізації квадратичної строго опуклої функції кількість ітерацій і, як наслідок, час виконання програми, зменшуються. Тому варіанти r-алгоритмів на основі прискореного розтягу простору в напрямках, наближених до напрямку нормованої різниці двох послідовних субградієнтів, можна вважати доволі перспективними, оскільки для деяких класів функцій вони можуть краще враховувати характер яружності функції та, при відповідній програмній реалізації, мати менший час виконання.

Ключові слова: оператор розтягу простору, $r(\alpha)$-алгоритм, кусково-лінійна функція.

UDC 519.85

Anton Suprun

Computational Experiments for the $r(\alpha)$-Algorithm with an Accelerated Implementation of Space Dilation

V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv
Correspondence: anton_s2007@ukr.net

In the article, a modification of the $r(\alpha)$-algorithm with adaptive line search which implements accelerated space dilation operation is considered. In classical $r(\alpha)$-algorithm with adaptive line search, space is dilated along the difference of the two successive subgradients. In its modified variant, space is dilated along the direction which is determined by the largest absolute value components of the vector of difference of the two successive subgradients. The number of such components varies at each iteration of the algorithm and depends on a control parameter. Thus, at each iteration only corresponding subspace of a smaller dimension is dilated and the number of multiplication operations used for this decreases from $2n^2 + 3n$ to $2nm + 2m + n$, where n is the dimension of the space, m is the number of non-zero components of the space dilation direction.

Description of the modified $r(\alpha)$-algorithm with adaptive line search which implements accelerated space dilation operation is given. Results of the numerical experiments of minimization of the two classes of convex piecewise linear functions and quadratic strongly convex functions are presented. To solve the problems C++ implementations of the classical $r(\alpha)$-algorithm with an adaptive line search ralgb5a and its modified version with accelerated space dilation ralgb5at were used. Obtained results show that for minimization of the piecewise linear functions compared with the classical $r(\alpha)$-algorithm its modified version requires approximately the same number of iterations to converge but uses a significantly smaller number of multiplication operations. For minimization of the quadratic strongly convex functions modified $r(\alpha)$-algorithm requires a smaller number of iterations to converge and as a result has a smaller execution time.

Therefore, it is concluded that the variants of the r-algorithm that implements accelerated space dilation operation along directions which approximate the difference of the two successive subgradients may be quite promising since for some classes of functions they might better take into account the shape of the function's ravines and with appropriate software implementation have a smaller execution time.

Keywords: space dilation operator, $r(\alpha)$-algorithm, piecewise linear function.