КІБЕРНЕТИКА та КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 519.85 DOI:10.34229/2707-451X.25.4.2

П.І. СТЕЦЮК, Н.І. ТУКАЛЕВСЬКА, О.М. ХОМ'ЯК, М.Г. СТЕЦЮК

R-АЛГОРИТМ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КВАДРАТИЧНОГО ПРОГРАМУВАННЯ

Описано розроблений алгоритм QPralg (Quadratic Programming by r-algoritm) та його програмну реалізацію мовою Octave для розв'язання задачі мінімізації опуклої квадратичної функції при лінійних двосторонніх обмеженнях. Він використовує метод негладких штрафних функцій та програму ralgb5a, яка реалізує варіант r-алгоритму з постійним коефіцієтом розтягу простору у напрямі різниці двох послідовних субградієнтів та адаптивним способом регулювання кроку в напрямі антисубградієнта в перетвореному просторі змінних. Алгоритм QPralg орієнтовано на випадок задач з невеликою кількістю змінних та дуже великою кількістю обмежень (від сотень тисяч до декількох мільйонів).

Ключові слова: *задача квадратичного програмування, робастна оптимізація, негладка штрафна функція, r-алгоритм, GNU Octave.*

Вступ. Задачі квадратичного програмування [1–3] відіграють важливу роль у сучасній оптимізації, особливо в умовах невизначеності, яка часто виникає в реальних прикладних задачах (від фінансового моделювання до інженерного проєктування та енергетичних систем). Робастний підхід [4] дозволяє отримати розв'язки, прийнятні за всіх допустимих реалізацій невизначеності, та приводить до задач великої розмірності.

Задача квадратичного програмування полягає у мінімізації опуклої квадратичної функції $Q(x)=1/2\,x^T Hx+c^T x$ при лінійних двосторонніх обмеженнях $b^{low}\le Ax\le b^{up}$, де $H\in\mathbb{R}^{n\times n}$ – невід'ємно визначена симетрична матриця, $c\in\mathbb{R}^n$ – вектор коефіцієнтів цільової функції, $x\in\mathbb{R}^n$ – вектор невідомих (змінних), $A\in\mathbb{R}^{m\times n}$ – матриця обмежень, $b^{low}\in\mathbb{R}^m$, $b^{up}\in\mathbb{R}^m$ – відповідно вектори нижніх та верхніх меж.

В статті представлено алгоритм QPralg (Quadratic Programming by r-algoritm), призначений для розв'язання задачі квадратичного програмування за допомогою *r*-алгоритму [5]. Алгоритм QPralg орієнтовано на задачі з невеликою кількістю змінних $(100\div300)$ та дуже великою кількістю лінійних двосторонніх обмежень (від кількох сотень тисяч до мільйонів).

Матеріал статті викладено у такій послідовності. У розділі 1 описано $r(\alpha)$-алгоритм з адаптивним способом регулювання кроку та наведено програму **ralgb5a** (на мові програмування Octave [6]), яка його реалізує. У розділі 2 наведено метод негладких штрафних функцій для задачі опуклого програмування та сформульовано дві теореми, що дають змогу обґрунтовувати скінченні значення штрафних коефіцієнтів. У розділі 3 описано алгоритм QPralg, його програмну реалізацію мовою Octave. У розділі 4 наведено результати обчислювальних експериментів для задач квадратичного програмування з двосторонніми обмеженнями.

1. Модифікація r(α)-алгоритму з адаптивним кроком. Для розв'язання задач нелінійного програмування можна використовувати метод негладких штрафних функцій разом з Octave-програмою ralgb5a, що реалізує *r*-алгоритм з адаптивним кроком – субградієтний метод з постійним коефіцієнтом розтягу простору у напрямі різниці двох послідовних субградієнтів та адаптивним способом регулювання кроку в напрямі антисубградієнта в перетвореному просторі змінних. Така модифікація відома як $r(\alpha)$-алгоритми [7], де α ($\alpha > 1$) – коефіцієнт розтягу простору, який залишається сталим на кожній ітерації *r*-алгоритмів.

Розглядається задача пошуку точки мінімуму опуклої функції $f(x)$, $x \in \mathbb{R}^n$. Мінімальне значення функції позначимо $f^* = f(x^*)$, де $x^* \in X^*$ – точка мінімуму. Нехай $\alpha > 1$ – коефіцієнт розтягу простору.

Означення. $r(\alpha)$-алгоритмом мінімізації функції $f(x)$ називається ітеративна процедура побудови послідовності n-вимірних векторів $\{x_k\}_{k=0}^{\infty}$ та послідовності $n \times n$-матриць $\{B_k\}_{k=0}^{\infty}$ за таким правилом:

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

де

$$\xi_k = \frac{B_k^T g_f(x_k)}{\left\| B_k^T g_f(x_k) \right\|}, \quad h_k \geq h_k^* = \arg\min_{h \geq 0} f(x_k - h B_k \xi_k), \tag{2}$$

$$\eta_k = \frac{B_k^T r_k}{\| B_k^T r_k \|}, \quad r_k = g_f(x_{k+1}) - g_f(x_k). \tag{3}$$

Тут x_0 – стартова точка, $B_0 = I_n$ – одинична $n \times n$-матриця[1], h_k^* – величина кроку до точки мінімуму функції $f(x)$, $R_\beta(\eta) = I_n + (\beta - 1)\eta\eta^T$ – оператор стиснення простору субградієнтів у нормованому напрямку η з коефіцієнтом $\beta = 1/\alpha < 1$, $g_f(x_k)$ і $g_f(x_{k+1})$ – субградієнти функції $f(x)$ у точках x_k та x_{k+1}. Якщо на ітерації k процесу (1) – (3) виконані умови зупинки, які наведемо нижче, то вважаємо $k^* = k$, $x_k^* = x_k$ і завершуємо роботу алгоритму.

Коментар. На кожній ітерації r-алгоритмів реалізується субградієнтний спуск для опуклої функції $\varphi(y) = f(B_k y)$ у перетвореному просторі змінних $y = A_k x$, де $A_k = B_k^{-1}$ – невироджена матриця. Дійсно, якщо обидві частини формули $x_{k+1} = x_k - h_k B_k \xi_k$ домножити зліва на матрицю A_k, отримаємо

$$y_{k+1} = A_k x_{k+1} = A_k x_k - h_k \xi_k = y_k - h_k \frac{B_k^T g_f(x_k)}{\left\| B_k^T g_f(x_k) \right\|} = y_k - h_k \frac{g_\varphi(y_k)}{\left\| g_\varphi(y_k) \right\|}, \tag{4}$$

де вектор $g_\varphi(y_k) = B_k^T g_f(x_k)$ є субградієнтом функції $\varphi(y) = f(B_k y)$ у точці $y_k = A_k x_k$ простору змінних $y = A_k x$.

[1] Як матрицю B_0 часто обирають діагональну матрицю D_n з додатними елементами на діагоналі, за допомогою якої здійснюється масштабування змінних.

Сімейство $r(\alpha)$-алгоритмів визначається коефіцієнтом розтягу простору $\alpha > 1$ та послідовністю величин кроків $\{h_k\}_{k=0}^{\infty}$, які використовуються для обчислення двох послідовних субградієнтів $g_f(x_k)$ і $g_f(x_{k+1})$. Розтягування за різницею цих субградієнтів (див. формулу (3)) дозволяє зменшити ступінь витягнутості функції у перетвореному просторі змінних. Вибір коефіцієнта $\alpha > 1$ та адаптація величин $\{h_k\}_{k=0}^{\infty}$ до умов зупинки визначають конкретні варіанти реалізації $r(\alpha)$-алгоритмів. Кількість ітерацій $r(\alpha)$-алгоритму, необхідна для знаходження точки x_{k^*}, що задовольняє умову $f(x_{k^*}) - f^* \le \varepsilon$, емпірично оцінюється як $k^* = O(n\log(1/\varepsilon))$, де n – кількість змінних.

В практичних реалізаціях $r(\alpha)$-алгоритмів зазвичай використовується адаптивний спосіб регулювання кроку. Він полягає у тому, що величина h_k налаштовується (адаптується) в процесі одновимірного спуску, який завершується, як тільки знайдено субградієнт, що утворює негострий кут із субградієнтом, який визначає напрямок спуску (умова завершення спуску за напрямком) [7]. Налаштування величини h_k здійснюється за допомогою чотирьох параметрів: $h_0 > 0$ – початкове значення кроку (використовується на першій ітерації, а надалі уточнюється), q_1 ($q_1 \le 1$) – коефіцієнт зменшення кроку (застосовується, якщо умова завершення виконується вже на першому кроці), q_2 ($q_2 \ge 1$) – коефіцієнт збільшення кроку. Через кожні n_h кроків одновимірного спуску ($n_h > 1$) величина кроку збільшується в q_2 раз. Якщо множина точок мінімуму X^* – обмежена, то після скінченної кількості ітерацій адаптивного спуску в напрямку нормованого антисубградієнта умова завершення спуску за напрямом обов'язково виконується [7].

Для зупинки ітераційного процесу використовуються параметри ε_x і ε_g. Алгоритм завершує роботу в точці $x_{k^*} \in [x_k, x_{k+1}]$, якщо виконується одна з таких умов: $\|x_{k+1} - x_k\| \le \varepsilon_x$ – зупинка за аргументом), або $\|g_f(x_{k^*})\| \le \varepsilon_g$ – зупинка за нормою градієнта (використовується для гладких функцій). Окрім того, застосовуються також стандартна зупинка, якщо перевищено задану максимальну кількість ітерацій **maxitn**, та аварійна зупинка, яка сигналізує про те, що функція $f(x)$ не є обмеженою знизу або початковий крок h_0 є надто малим і потребує збільшення.

Програмну реалізацію $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за формулами (1) – (3) здійснено у вигляді Octave-функції **ralgb5a** [8, 9]. Назва "**b5a**" вказує на те, що в основу реалізації покладено r-алгоритм у B-формі, в якій на кожній ітерації коригується $n \times n$-матриця B. Кожна ітерація потребує $5n^2$ арифметичних операцій множення, що визначає обчислювальну складність одного кроку методу з адаптивним регулювання кроку. При цьому фіксуються два найбільш часто використовувані параметри $q_2 = 1.1$ і $n_h = 3$. Крім того, у функції **ralgb5a** передбачено параметр **intp (int**erval for **p**rint**)**, який відповідає за виведення інформації про перебіг процесу мінімізації через кожні **intp** ітерацій. Це дає змогу суттєво обсяг виведеної інформації у випадках, коли розв'язуються задачі з сотнями або тисячами змінних, а загальна кількість ітерацій оцінюється тисячами і десятками тисяч.

Якщо ітераційний процес запускається зі стартової точки x_0, то параметри $r(\alpha)$-алгоритму рекомендується вибирати наступними: $\alpha \in [2,4]$, $q_1 = 1.0$ (для негладких функцій), $q_1 = 0.8 \div 0.95$ (для гладких функцій), $h_0 \approx \|x_0 - x^*\|$ – оцінка відстані від стартової точки x_0 до точки мінімуму

x^*. Як правило, використовуються такі параметри зупинки: $\varepsilon_x \approx 10^{-6}$, $\left\|x_r - x^*\right\|$, **maxitn** $\approx 20n$**.** Тут параметр ε_g використовується для гладких функцій, а параметр ε_x – для негладких функцій. При правильному виборі параметрів α, h_0 та q_1 можна значно скоротити кількість ітерацій для виконання одних і тих самих критеріїв зупинки ε_x та ε_g. Це залежить від конкретного виду функції, що мінімізується, ступеня її яружності та масштабу змінних.

У статті [9] Octave-функція **ralgb5a** доповнена параметром **im**: якщо **im = 1**, то x_r^* – наближення до точки мінімуму опуклої функції $f(x)$, якщо **im = –1**, то x_r^* – наближення до точки максимуму увігнутої функції $f(x)$. Вона використовує Octave-функцію **function [f, g] = calcfg (x)**, яка обчислює значення функції $f = f(x)$ та її субградієнта (суперградієнта) $g = g_f(x)$ у точці x. Функція визначається користувачем і може мати будь-яке ім'я, допустиме відповідно до синтаксичних правил Octave.

Код модернізованої Octave-функції **ralgb5a**, який включає короткі англомовні коментарі для вхідних та вихідних параметрів наведено у додатку А.

2. **Метод негладких штрафних функцій.** Метод штрафних функцій із негладкою функцією штрафу застосовується для врахування обмежень у вигляді рівностей або нерівностей в умовних задачах математичного програмування. Цей метод дозволяє звести умовну задачу математичного програмування на мінімум до задачі безумовної мінімізації негладкої функції. Розглянемо метод негладких штрафних функцій відповідно до монографії [5, стор. 188–189], який базується на результатах роботи [10].

Нехай розглядається задача опуклого програмування

$$\min_{x \in R^n} f_0(x) \quad \text{при} \quad f_i(x) \le 0, \quad i = 1,\ldots,m. \tag{5}$$

Визначимо функцію

$$S(x) = f_0(x) + \sum_{i=1}^{m} p_i\left[f_i(x)\right], \tag{6}$$

де $p_i(t)$, $i = 1,\ldots,m$, задовольняють умови опуклості, $p_i(t) = 0$ для $t < 0$, $p_i(t) \ge 0$ для $t > 0$. Допустимо, що існує x^* – оптимальний розв'язок задачі $\inf\limits_x S(x)$.

Теорема 1. Для того, щоб x^* було оптимальним розв'язком задачі (5), необхідно, щоб

$$\lim_{t \to +0} \frac{p_i(t)}{t} \ge \overline{y}_i, \quad i = 1,\ldots,m,$$

де $\overline{y} = (\overline{y}_1,\ldots,\overline{y}_m)$ – деякі множники Лагранжа задачі (5).

Для того, щоб (5) і (6) мали однакову множину оптимальних розв'язків, достатньо, щоб $\lim\limits_{t \to 0} \dfrac{p_i(t)}{t} > \overline{y}_i$.

Найпростіший варіант негладкої функції штрафу має вигляд

$$p(t) = \begin{cases} 0, & t \le 0, \\ ct, & t > 0. \end{cases} \tag{7}$$

З теореми 1 випливає, що для того, щоб задача мінімізації $S(x)$ була еквівалентна задачі (5) при

$$p_i(t) = \begin{cases} 0, & t \le 0, \\ c_i t, & t > 0. \end{cases} \quad i = 1, \ldots, m, \tag{8}$$

достатньо вибрати $c_i > \overline{y_i}$.

Особливий інтерес становить штрафна функція

$$T(x) = f_0(x) + p\left[\max_{i \in 1,m} f\left(x_i\right)\right],$$

де $p(t)$ обчислюється за формулою (7). Очевидно, що за умови $c > \sum_{i=1}^{m} \overline{y_i}$ виконується

$$p\left(\max_{i \in 1,m} f\left(x_i\right)\right) \ge \sum_{i=1}^{m} p_i\left[f_i(x)\right],$$

де p_i визначається за формулами (8), $c = \sum_{i=1}^{m} c_i$; $c_i > \overline{y_i}$. Таким чином, при $c > \sum_{i=1}^{m} \overline{y_i}$ задача мінімізації функції $T(x)$ еквівалентна задачі (5).

При використанні для розв'язання спеціальних умовних задач математичного програмування негладкої штрафної функції у формі функції максимуму можна застосовувати теорему Б.М. Пшеничного про точні штрафні функції [11, теорема 2.14, стор. 25].

Нехай для врахування обмежень у задачі (5) використовується негладка штрафна функція у формі функції максимуму:

$$\Phi_N(x) = f_0(x) + N \times \max\{0, f_1(x), \ldots, f_m(x)\}. \tag{9}$$

Розглянемо сімейство параметричних задач

$$V(z) = \inf\left\{ f_0(x) : f_i(x) \le z_i; i = 1, \ldots, m \right\},$$

залежне від вектора $z \in R^m$. Очевидно, що

$$V(0) = \inf\left\{ f_0(x) : f_i(x) \le 0; i = 1, \ldots, m \right\}$$

збігається з розв'язком задачі (5). Справедлива наступна теорема [11].

Теорема 2. Нехай $\inf\limits_{\lambda > 0} \dfrac{V(\lambda e) - V(0)}{\lambda} = -L > -\infty$, де e – m-вимірний вектор, усі компоненти якого дорівнюють одиниці. Якщо $N > L$, то тоді точки мінімуму задач $V(0)$ і $\inf\limits_{x \in M} \Phi_N(x)$, де $\Phi_N(x)$ визначається за формулою (9), співпадають.

Теорема 2 дає інструмент для встановлення точного значення штрафного множника при використанні для розв'язання задачі (5) негладкої штрафної функції у формі функції максимуму. Продемонструємо його для задачі нелінійного програмування [12], яка має такий вигляд

$$F^* = \max_x \sum_{j=1}^{n} c_j x_j \ , \tag{10}$$

$$\sum_j a_{ij} x_j + \Omega \sqrt{\sum_{j \in J_i} \hat{a}_{ij}^2 x_j^2} \le b_i, \quad i = 1, \ldots, m, \tag{11}$$

$$x_j \ge 0, \quad j = 1, \ldots, n. \tag{12}$$

Якщо усі b_i, $i=1,...,\mathrm{m}$ є додатними, то задача (10) – (12) може бути записана в еквівалентній формі:

$$-F^* = \min_x \sum_{j=1}^{n} -c_j x_j \,, \tag{13}$$

$$\frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2 x_j^2} \le 1, \quad i=1,...,m, \tag{14}$$

$$x_j \ge 0, \quad j=1,...,n. \tag{15}$$

Нехай для врахування обмеження (14) використовується негладка штрафна функція у формі функції максимуму:

$$\Phi_N(x) = \sum_{j=1}^{n} -c_j x_j + N\max\left\{0, \max_{i=1,...,m}\left\{\frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2 x_j^2} - 1\right\}\right\}, \tag{16}$$

де $N>0$ – штрафний параметр.

Теорема 3. При $N > F^*$ задача (13) – (15) еквівалентна задачі

$$F^* = \min_{x\ge 0}\Phi_N(x). \tag{17}$$

Доведення. Обчислимо L, яке згідно з теоремою 2 визначає скінченне значення штрафного параметра $N>L$, при якому задача (16) є еквівалентною задачі (13) – (15). Для цього розглянемо сімейство параметричних задач

$$V(t) = \inf_x\left\{\sum_{j=1}^{n} -c_j x_j : \frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2 x_j^2} \le 1+t, i=1,...,m,\ x_j\ge 0,\ j=1,\ldots,n\right\},$$

залежне від параметра $t\in R$. Очевидно, що

$$V(0) = \inf_x\left\{\sum_{j=1}^{n} -c_j x_j : \frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2 x_j^2} \le 1, i=1,...,m,\ x_j\ge 0,\ j=1,\ldots,n\right\}$$

співпадає з розв'язком задачі (13) – (15), звідки $V(0) = -F^*$.

Сімейство задач $V(t)$ при $t>0$ можна записати у такий спосіб:

$$V(t) = \inf_x\left\{\sum_{j=1}^{n} -c_j x_j : \frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2 x_j^2} \le 1+t, i=1,...,m,\ x_j\ge 0,\ j=1,\ldots,n\right\} =$$

$$= \inf_x\left\{\sum_{j=1}^{n} -c_j \frac{x_j}{1+t} : \frac{1}{b_i}\sum_j a_{ij}x_j + \frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2\left(\frac{x_j}{1+t}\right)^2} \le 1, i=1,...,m, \frac{x_j}{1+t}\ge 0,\ j=1,\ldots,n\right\},$$

звідки, зробивши заміну змінних $\tilde{x}_j = \dfrac{x_j}{1+t}$, маємо

$$V(t)=(1+t)\inf_{\tilde{x}}\left\{\sum_{j=1}^{n}-c_j\tilde{x}_j:\frac{1}{b_i}\sum_j a_{ij}\tilde{x}_j+\frac{\Omega}{b_i}\sqrt{\sum_{j\in J_i}\hat{a}_{ij}^2\tilde{x}_j^2}\le 1+t, i=1,...,m,\ \tilde{x}_j\ge 0,\ j=1,\ldots,n\right\}=(1+t)V(0).$$

У результаті отримуємо

$$-L=\inf_{t>0}\frac{V(t)-V(0)}{t}=\inf_{t>0}\frac{(1+t)-1}{t}V(0)=V(0),$$

звідки, з урахуванням того, що $V(0)=-F^*$, маємо $L=-V(0)=F^*$. *Теорема доведена.*

Використання негладких штрафних функцій приводить до задач мінімізації функцій яружного виду. Для їх розв'язання можна використовувати ефективний варіант r-алгоритму з адаптивним кроком, який описано в розділі 1.

3. Алгоритм QPralg для задачі квадратичного програмування. Задача квадратичного програмування полягає у мінімізації опуклої квадратичної функції $Q(x)=\frac{1}{2}x^THx+c^Tx$ при лінійних двосторонніх обмеженнях $b^{low}\le Ax\le b^{up}$, де $H\in\mathbb{R}^{n\times n}$ – невід'ємно визначена симетрична матриця, $c\in\mathbb{R}^n$ – вектор коефіцієнтів цільової функції, $x\in\mathbb{R}^n$ – вектор невідомих (змінних), $A\in\mathbb{R}^{m\times n}$ – матриця обмежень, $b^{low}\in\mathbb{R}^m$, $b^{up}\in\mathbb{R}^m$ – вектори лівих (нижніх) та правих (верхніх) частин.

Формулювання задачі квадратичного програмування з двосторонніми обмеженнями має такий вигляд: знайти

$$Q^*=Q(x^*)=\min_{x\in\mathbf{R}^n}\left\{Q(x)=\frac{1}{2}x^THx+c^Tx\right\},\tag{18}$$

при обмеженнях

$$b_i^{low}\le\sum_j a_{ij}x_j\le b_i^{up},\ i=1,\ldots,m.\tag{19}$$

Тут лінійні нерівності (19) є двосторонніми обмеженнями (нерівностями). Якщо матриця $H\equiv 0$, то задача (18) – (19) переходить в задачу лінійного програмування з двосторонніми обмеженнями [13, 14].

Зауважимо, що цільова функція (18) визначена так, що у ній відсутній вільний член. Він не відіграє ніякої ролі при побудові алгоритмів розв'язання задач квадратичного програмування, але є досить зручним для представлення самого значення цільової функції, наприклад, $\|x-x_0\|^2=x^Tx-2x^Tx_0+x_0^Tx_0$. Обмеження (18) є двосторонніми лінійними обмеження (нерівностями).

Задача (18) – (19) у наведеній формі, де обмеження (19) є двосторонніми лінійними, часто є незручною для реалізації алгоритмів розв'язання задач квадратичного програмування. Тому розглянемо її еквівалентну форму, задачу квадратичного програмування у такому вигляді: знайти

$$Q^*=\min_{x\in\mathbb{R}^n}\left\{\frac{1}{2}x^THx+c^Tx\right\},\tag{20}$$

при обмеженнях

$$\sum_j a_{ij}x_j\le b_i^{up},\ i=1,\ldots,m,\tag{21}$$

$$-\sum_j a_{ij}x_j\le -b_i^{low},\ i=1,\ldots,m.\tag{22}$$

Задачу (20) – (22) також можна розглядати як задачу квадратичного програмування з двосторонніми обмеженнями: знайти

$$Q^* = \min_{x\in\mathbb{R}^n}\left\{\frac{1}{2}x^T Hx + c^T x\right\}, \tag{20'}$$

при обмеженнях

$$-1000 \le \sum_j a_{ij}x_j \le b_i^{up},\ i=1,\ldots,m, \tag{21'}$$

$$-1000 \le -\sum_j a_{ij}x_j \le -b_i^{low},\ i=1,\ldots,m \tag{22'}$$

для явно занижених значень нижніх границь.

Алгоритм QPralg використовує негладку штрафну функцію у формі функції максимуму для врахування двосторонніх обмежень (19). Він полягає у розв'язанні задачі безумовної мінімізації негладкої штрафної функції

$$Q_P(x) = \frac{1}{2}x^T Hx + c^T x + P * \max\left\{0, \max_{i=1,\ldots m}\left\{\sum_j a_{ij}x_j - b_i^{up}\right\}, \max_{i=1,\ldots m}\left\{b_i^{low} - \sum_j a_{ij}x_j\right\}\right\}, \tag{23}$$

де $P>0$ – штрафний коефіцієнт. Ця задача розв'язується за допомогою octave-програми ralgb5a. Нехай x^* – оптимальний розв'язок задачі мінімізації функції (23).

Теорема 4. Для того, щоб x^* було оптимальним розв'язком задачі (18) – (19), необхідно, щоб $P \ge \sum_{i=1}^{2m}\lambda_i^*$, де $\lambda_1^* \ge 0,\ldots,\lambda_{2m}^* \ge 0$ – оптимальні множники Лагранжа для задачі (20) – (22).

Теорема 4 – наслідок застосування теореми 1 до задачі (20) – (22).

Обчислення значення функції $Q_P(x)$ та її субградієнта реалізує octave-функція fgQP(x), яка має такий вигляд:

```
function [f,g] = fgQP(x)
global H c A blow bup n m P2
f = 0.5*x'*H*x + sum(c.*x); g = H*x + c;
tmp1 = A*x; tmp2 = [blow - tmp1; tmp1 - bup];
[tmpmax imax] = max(tmp2);
if (tmpmax >= 0.d0)
  if (imax <= m)
    f = f + P2*tmpmax;
    g = g - P2*A(imax,1:n)';
  endif
  if (imax > m)
    f = f + P2*tmpmax;
    g = g + P2*A(imax-m,1:n)';
  endif
endif
endfunction #fgQP
```

Вектор невідомих (змінних) задачі передається як формальний параметр octave-функції fgQP(x), вхідні дані задачі передаються через загальну пам'ять як глобальні змінні та є наступними:

- H – невід'ємно визначена симетрична матриця,

- `c` – вектор коефіцієнтів цільової функції,
- `A` – матриця обмежень,
- `blow, bup` – вектори лівих та правих частин,
- `n` – кількість змінних,
- `m` – кількість обмежень,
- `P2` – штрафний коефіцієнт.

Вибрати скінченний штрафний коефіцієнт $P2 = P$ дозволяє теорема 4.

В частковому випадку задача квадратичного програмування переходить в задачу лінійного програмування, для окремих випадків якої можна обґрунтувати вибір штрафного коефіцієнта [9].

Розглянемо ЛП-задачу загального вигляду у такому формулюванні:

$$c^* = \min_{x\in\mathbb{R}^n} \sum_{j=1}^n c_j x_j \quad \text{при обмеженнях} \quad \sum_{j=1}^n a_{ij} x_j \le b_i, \quad i = 1,\ldots,m, \tag{24}$$

та її частковий випадок у такому формулюванні:

$$c_1^* = \min_{x\in\mathbb{R}^n} \sum_{j=1}^n c_j x_j \quad \text{при обмеженнях} \quad \sum_{j=1}^n a_{ij} x_j \le 1, \quad i = 1,\ldots,m. \tag{25}$$

Штрафна функція у формі функції максимуму для ЛП-задачі (24) має вигляд:

$$c_P(x) = c^T x + P \cdot \max\left\{0, \max_{i=1,\ldots m}\left\{\sum_{j=1}^n a_{ij} x_j - b_i\right\}\right\}, \tag{26}$$

а аналогічна штрафна функція для ЛП-задачі (25) має такий вигляд:

$$c_N(x) = c^T x + N \cdot \max\left\{0, \max_{i=1,\ldots m}\left\{\sum_{j=1}^n a_{ij} x_j - 1\right\}\right\}, \tag{27}$$

де $P > 0$ та $N > 0$ – штрафні коефіцієнти.

Для ЛП-задачі (24) оптимальні множники Лагранжа $\lambda_1^* \ge 0, ..., \lambda_m^* \ge 0$, які відповідають обмеженням $\sum_{j=1}^n a_{ij} x_j \le b_i$, $i = 1,\ldots,m,$ є розв'язком ЛП-задачі

$$c^* = \max_{\lambda\in\mathbb{R}^m, \lambda\ge 0} \left(-\sum_{i=1}^m b_i \lambda_i\right) \quad \text{при обмеженнях} \quad \sum_{i=1}^m a_{ij}\lambda_i = -c_j, \quad j = 1,\ldots,n. \tag{28}$$

Теорема 5 [15]. *Якщо* $P > P_* = \sum_{i=1}^m \lambda_i^*$, то ЛП-задача (24) – еквівалентна задачі безумовної оптимізації $c^* = c_P^* = \min_{x\in\mathbb{R}^n} c_P(x)$, де $c_P(x)$ визначається формулою (26).

Теорема 5 – наслідок застосування теореми 1 до ЛП-задачі (24).

Теорема 6 [15]. *Якщо* $N > L = \left|c_1^*\right|$, то ЛП-задача (25) – еквівалентна задачі оптимізації $c_1^* = c_N^* = \min_{x\in\mathbb{R}^n} c_N(x)$, де $c_1^* < 0$, а $c_N(x)$ визначається формулою (27).

Теорема 6 – наслідок застосування теореми 2 до ЛП-задачі (25).

4. Обчислювальні експерименти. Результати обчислювальних експериментів для програми QPralg включають тестування на двох сімействах задач квадратичного програмування. Перше сімейство задач характеризується малою кількістю змінних (до двадцяти) та дуже великою кількістю двосторонніх обмежень (десятки мільйонів). Друге сімейство задач має невелику кількість змінних ($n = 100 \div 300$) та велику кількість двосторонніх обмежень ($m = 50000 \div 150000$).

Для першого сімейства задача полягає у знаходженні точки мінімальної норми із допускової множини інтервальної системи лінійних рівнянь [16], яка забезпечує існування такого значення інтервальної правої частини, що система має розв'язок при будь-яких значеннях інтервальних коефіцієнтів лінійних форм. Для інтервальної системи $Ax = b$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^k$ допускова множина описується за допомогою $k * 2^n$ двосторонніх лінійних нерівностей (обмежень), кількість яких дуже швидко збільшується із збільшенням n. Так, наприклад, для $n = 20$ (кількість змінних) та $k = 15$ (кількість рівнянь) маємо 15 мільйонів 728 тисяч 640 двосторонніх лінійних обмежень.

Проблему знаходження точки мінімальної норми із допускової множини інтервальної системи лінійних рівнянь можна сформулювати як задачу квадратичного програмування такого вигляду: знайти

$$f_1^* = \min \sum_{i=1}^{n} x_i^2 , \tag{29}$$

при обмеженнях

$$\underline{b}_i \le \sum_j a_{ij} x_j \le \overline{b}_i , \; i = 1,\ldots,k , \; \forall a_{ij} \in \left\{ \underline{a}_{ij}, \overline{a}_{ij} \right\}, \tag{30}$$

де $\underline{a}_{ij}$, $\overline{a}_{ij}$, $\underline{b}_i$, $\overline{b}_i$ – грані відповідних інтервальних об'єктів $b = [\underline{b}_i, \overline{b}_i]$ та $a_{ij} = [\underline{a}_{ij}, \overline{a}_{ij}]$. Цільова функція (29) означає мінімізацію квадрата норми вектора, який задовольняє двостороннім лінійним обмеженням (30), кількість яких у випадку, коли інтервальними є усі коефіцієнти матриці A, є рівною $k * 2^n$. Octave-код програми для розв'язання задачі (29) – (30) наведено в додатку Б.

Задача (29) – (30) сформульована у термінах двосторонніх лінійних обмежень. Тому, як альтернативу задачі (29) – (30) будемо розглядати задачу квадратичного програмування у такому формулюванні: знайти:

$$f_1^* = \min \sum_{i=1}^{n} x_i^2 \tag{31}$$

при обмеженнях

$$\sum_j a_{ij} x_j \le \overline{b}_i , \; i = 1,\ldots,k , \; \forall a_{ij} \in \left\{ \underline{a}_{ij}, \overline{a}_{ij} \right\}, \tag{32}$$

$$-\sum_j a_{ij} x_j \le -\underline{b}_i , \; i = 1,\ldots,k , \; \forall a_{ij} \in \left\{ \underline{a}_{ij}, \overline{a}_{ij} \right\}. \tag{33}$$

Задача (31) – (33) включає у два рази більше обмежень, ніж задача (29) – (30). Її легко звести до потрібного для програми QPralg вигляду задачі квадратичного програмування з двосторонніми обмеженнями, враховуючи що нижні границі обмежень (32) та (33) є рівними $-\infty$.

Для першого сімейства задач було проведено два обчислювальні експерименти. Мета першого експерименту це оцінка часу розв'язання задачі (29) – (30) при $n = 20$ та $k = 15$ на комп'ютері з процесором CPU 2.90GHz. При цьому для зберігання матриці обмежень використовується майже 3 ГБ оперативної пам'яті ($8 * 15 * 2^{(20+1)} \approx 8 * 300\text{млн} = 2,4\text{ГБ}$). Такий обсяг пам'яті наближається до граничного для комп'ютерів з 8 ГБ оперативної пам'яті, враховуючи, що операційна система Windows займає близько половини оперативної пам'яті.

Час, витрачений на розв'язання задачі, становить 356.00 секунд. Це свідчить про те, що задачі квадратичного програмування з невеликою кількістю змінних (до двадцяти) та дуже великою кількістю двосторонніх обмежень (порядку десятків мільйонів) можуть бути розв'язані на сучасних

комп'ютерах за декілька хвилин. Знайдений оптимальний розв'язок містить 6976 активних обмежень, що пояснюється тим, що коефіцієнти матриці двосторонніх обмежень генерувалися у діапазоні від 0 до 9, і багатьом однаковим значенням параметрів $\underline{a}_{ij}$ та $\overline{a}_{ij}$ відповідали по дві однакові двосторонні нерівності.

Мета другого експерименту це порівняння часових витрат програм QPralg та Gurobi [17] під час розв'язання задачі (31) − (33) для наборів параметрів $n=20$ та $k=1$, $n=19$ та $k=2$, $n=18$ та $k=4$, $n=17$ та $k=8$. Такі розміри задач було обрано з огляду на те, що під час розв'язання задачі (31) − (33) для параметрів $n=20$ та $k=2$ програмі Gurobi не вистачало обсягу оперативної пам'яті, що виділяється NEOS-сервером [18] (за замовчуванням – 3ГБ, $m=2*2^{20+1}=2^{22}=4194304$). Часові витрати програм QPralg та Gurobi для вибраних значень n та k наведено в табл. 1, де в останній колонці у дужках вказано час розв'язання задачі (29) – (30), яка має удвічі менше обмежень, ніж задача (31) – (33).

ТАБЛИЦЯ 1. Затрати часу Gurobi (t_1) та QPralg (t_2) для задач (31) – (33)

№	n	k	m	t_1 (сек.)	t_2 (сек.)
1	20	1	2097152	221.015	17.53 (7.20)
2	19	2	2097152	94.8594	16.23 (8.29)
3	18	4	2097152	50.7674	16.25 (8.21)
4	17	8	2097152	31.4103	14.75 (7.44)

З даної таблиці видно, що для розв'язання задачі (31) − (33) при $n=20$ та $m=1$ програма QPralg затратила в 10 разів менше часу ніж програма Gurobi, для розв'язання задачі (31) − (33) при $n=17$ та $m=8$ затрати є меншими у 2 рази. З четвертої колонки табл. 1 бачимо, що затрати програми Gurobi зменшуються у два рази для кожної наступної задачі, що може пояснюватися тим, що для кожної із задач усі рівняння вибирались однаковими, а це означає, що для другої задачі генерується 2 групи однакових обмежень (32) та (33), для третьої задачі генерується 4 групи однакових обмежень, для четвертої задачі − 8 груп однакових обмежень. Затрати програми QPralg мало відрізняться у залежності від номеру задачі, що пов'язано з тим, що для всіх задач обчислення виконуються з матрицями обмежень, які мають однакові розміри.

Друге сімейство задач квадратичного програмування пов'язане з проєктуванням деякої заданої точки $z\in R^n$ на поліедр [19], який задається системою лінійних двосторонніх обмежень. У цьому випадку задача квадратичного програмування має такий вигляд: знайти

$$f_2^*=\min\sum_{i=1}^{n}\left(x_i-z_i\right)^2 \tag{34}$$

при обмеженнях

$$-0.1\le\sum_j a_{ij}x_j\le 0.1,\ i=1,\ldots,m, \tag{35}$$

де коефіцієнти a_{ij} обчислюються за формулою $a_{ij}=\tilde{a}_{ij}-\frac{1}{n}\sum_{j=1}^{n}\tilde{a}_{ij}$, значення $\tilde{a}_{ij}$ генеруються випадковим чином з діапазону $[0,5]$. Для всіх $i=1,\ldots,m$ виконуються рівності $\sum_{j=1}^{n}a_{ij}=0$, тому точки, всі

компоненти яких є однаковими, задовольняють системі обмежень (35). Це буде мати місце і тоді, коли нижня та верхня границі двосторонніх нерівностей є рівними нулю.

Мета експерименту це оцінка часу роботи програми QPralg для задачі (34) − (35) за трьома варіантами її розмірів: $n=300$ та $m=50000$, $n=200$ та $m=75000$, $n=100$ та $m=150000$. Розглядалися два варіанти точок z, які проєктуються на поліедр. У першому випадку проєктувалася випадкова точка з компонентами із діапазону $[0,5]$, а у другому – точка $(1,1,\ldots,1)^T$. Зведені результати розрахунків для вибраних значень n та m наведено в табл. 2, де c^* – знайдене мінімальне значення цільової функції, а fr – мінімальне значення штрафної функції з коефіцієнтом $P_2=100$.

ТАБЛИЦЯ 2. Часові затрати QPralg для двох варіантів задачі (34) – (35)

Задача (34) – (35) (z − випадкова точка)				
n	m	time(сек.)	c^*	fr
300	50000	59.27	–957.7919	–957.7919
200	75000	68.20	–685.6359	–685.6359
100	150000	101.41	–328.4385	–328.4385
Задача (34) – (35) ($z=(1,1,\ldots,1)^T$)				
n	m	time(сек.)	c^*	fr
300	50000	17.34	–300.0000	–300.0000
200	75000	13.64	–200.0000	–200.0000
100	150000	5.75	–100.0000	–100.0000

У випадку, коли проєктувалася випадкова точка, програма QPralg затратила значно більше часу, тому що розв'язки задач досягаються у граничних точках поліедра, заданого обмеженнями (35). Для другого випадку розв'язок задачі знаходиться всередині поліедра, що зумовило менші затрати часу. З четвертої та п'ятої колонки табл. 2 видно, що мінімальне значення цільової функції та мінімальне значення штрафної функції співпадають з точністю до всіх значущих цифр. В задачі (34) – (35) цільова функція представлена таким чином, що для проєкції точки $z=(1,1,\ldots,1)^T$ мінімальне значення цільової функції дорівнює кількості змінних зі знаком мінус. За цим критерієм можна оцінити точність розв'язання задачі для другої точки, що проєктується.

Час, затрачений на розв'язання задачі, не більше декількох хвилин, що означає, що задачі квадратичного програмування з малою кількістю змінних (до 300) та великою кількістю двосторонніх обмежень (порядку декількох десятків тисяч) на сучасних комп'ютерах можна розв'язувати за декілька хвилин.

Висновки. Квадратичні задачі оптимізації дозволяють точно моделювати широкий спектр прикладних процесів: від управління динамічними системами до фінансового планування та машинного навчання. В реальних умовах параметри таких моделей часто є неточними або змінними, що може істотно впливати на якість отриманих розв'язків. Робастні підходи забезпечують стійкість розв'язків до невизначеностей, дозволяючи отримувати надійні рішення, прийнятні за всіх допустимих реалізацій вхідних даних.

У даній роботі наведено опис розробленого алгоритму **QPralg** (**Q**uadratic **P**rogramming by **r**–**alg**oritm) та його програмну реалізацію мовою Octave для знаходження розв'язку задачі мінімізації опуклої квадратичної функції при лінійних двобічних обмеженнях. Вони використовують метод негладких штрафних функцій та octave–програму **ralgb5a**, яка реалізує *r*–алгоритм з адаптив-

ним кроком – субградієтний метод з постійним коефіцієтом розтягу простору у напрямі різниці двох послідовних субградієнтів та адаптивним способом регулювання кроку в напрямі антисубградієнта в перетвореному просторі змінних.

Алгоритм **QPralg** орієнтовано на випадок задач з невеликою кількістю змінних $(100 \div 300)$ та дуже великою кількістю обмежень (від сотень тисяч до декількох мільйонів). Розв'язання відповідних задач квадратичного програмування за допомогою комерційного солвера **Gurobi** вимагає у декілька разів більше часу для сучасних комп'ютерів з обсягом оперативної пам'яті 8 Гб. Велика кількість обмежень дозволяє розв'язувати квадратичну та лінійну задачі робастної оптимізації для значної кількості реалізацій вектора $\xi \in \mathbb{R}^m$ із множини невизначеності.

Подяка. Робота підтримана проєктами 2.3/25–П НАН України та № 2М–2024 (0124U002162) ДО «ВЦП КНУ ім. Т. Шевченка при НАН України».

Авторські внески. Стецюк П.І. – концептуалізація, керівництво; Тукалевська Н.І. – методологія, дослідження, редагування; Хом'як О.М. – узагальнення, написання, оригінальна чернетка; Стецюк М.Г. – програмне забезпечення, написання.

Додаток А. Код Octave–функції ralgb5a

```
#   Octave-function ralgb5a                                          #rowc01
# Input parameters:                                                  #rowc02
#   calcfg -- name of the function calcfg(x) for calculation       #rowc03
#             of f(x) and its sub(super)gradient g(x),             #rowc04
#   im -- minimize(im = 1), maximize(im = -1),                       #rowc05
#   x -- starting point (it is modified in the program), x(1:n)    #rowc06
#   alpha -- coefficient of space dilation (alpha = 2:4)           #rowc07
#   h0, q1 -- parameters of the adaptive step adjustment,          #rowc08
#   recommend: h0=1, q1=1.0 - nonsmooth, q1=0.8:0.95 - smooth      #rowc09
#   epsx, epsg, maxitn -- stop parameters                          #rowc10
#   intp -- print information for every intp iteration             #rowc11
# Output parameters:                                               #rowc12
#   xr -- record point (with the best function value), xr(1:n)     #rowc13
#   fr -- the value of the function f at the point xr              #rowc14
#   itn -- the number of iterations                                #rowc15
#   nfg -- the number of function calcfg calls                     #rowc16
#   ist -- exit code: 2-epsg, 3-epsx, 4-maxitn, 5-warning          #rowc17
# For more details see:  Stetsyuk, P.I. Theory and Software        #rowc18
#   Implementations of Shor's r-Algorithms. Cybernetics and        #rowc19
#   Systems Analysis 53, 692-703 (2017)                            #rowc20
#
function [xr,fr,itn,nfg,ist] = ralgb5a(calcfg,im,x,alpha,h0,q1,  #row001
                                       epsg,epsx,maxitn,intp),   #row002
itn = 0, B = eye(length(x)), hs = h0, lsa = 0, lsm = 0,          #row003
xr = x, [fr,g0] = calcfg(xr), nfg = 1,                           #row004
if (intp>0)                                                      #row005
  printf("itn%5d  f%15.6e  fr%15.6e  nfg%5d\n",itn,fr,fr,nfg),   #row006
endif                                                            #row007
if(norm(g0) < epsg) ist = 2,  return, endif                      #row008
for (itn = 1:maxitn)                                             #row009
   dx = B * (g1 = B' * g0)/norm(g1),                             #row010
   d = 1, ls = 0, ddx = 0,                                       #row011
```

```
      while (d > 0)                                                    #row012
         x -= im*hs * dx, ddx += hs * norm(dx),                        #row013
         [f, g1] = calcfg(x), nfg ++,                                  #row014
         if (im*f < im*fr)  fr = f, xr = x,  endif                     #row015
         if(norm(g1) < epsg) ist = 2,  return, endif                   #row016
         ls ++, (mod(ls,3) == 0) && (hs *= 1.1),                       #row017
         if(ls > 500) ist = 5,  return, endif                          #row018
         d = dx' * g1,                                                 #row019
      endwhile                                                         #row020
      (ls == 1) && (hs *= q1), lsa=lsa+ls, lsm=max(lsm,ls),            #row021
      if(mod(itn,intp)==0)                                             #row022
        if (intp>0)                                                    #row023
          printf("itn %4d  f %14.6e  fr %14.6e", itn, f, fr),          #row024
          printf("  nfg %4d  lsa %3d  lsm %3d\n", nfg, lsa, lsm),      #row025
        endif                                                          #row026
        lsa=0, lsm=0,                                                  #row027
      endif                                                            #row028
      if(ddx < epsx) ist = 3,  return, endif                           #row029
      xi = (dg = B' * (g1 - g0) )/norm(dg),                            #row030
      B += (1 / alpha - 1) * B * xi * xi',                             #row031
      g0 = g1,                                                         #row032
   endfor                                                              #row033
   ist = 4,                                                            #row034
   endfunction                                                         #row035
```

Додаток Б. Octave–код програми для розв'язання задачі (29) – (30)

```
# Code for solve_time of QPralg for problem (29) - (30): n=20, m=15
global H c A blow bup n m pen1

printf("\n"); # set parameters for r-algorithm
im = 1, alpha = 4.0, h0 = 1.0, q1 = 0.95,
epsx = 1.e-9, epsg = 1.e-8, maxitn = 20000, intp = 100,
pen1 = 10,

rand("seed", 2020);

tstart=time();
printf("\n");
nA = 20, mA = 15,
rand("seed", 2020);
A0 = round(8.5*rand(mA,nA));
x0 = ones(nA,1); rhs = A0*x0;
A_low = A0 - 0.1*abs(A0); A_up = A0 + 0.1*abs(A0);
printf("\n");
A_low = round(A_low), A_up = round(A_up),
diff_A = A_up-A_low,
b_low = rhs - 0.2*abs(rhs), b_up = rhs + 0.2*abs(rhs),

#  All binary Codes of length n
printf("\n");
n = nA;
iCode1 = zeros(2**n,n);
tic(),
```

```
for i=0:2**n-1
   nc1 = i;
   for j=0:n-1
     nc2 = round(nc1/2.0-0.25);
     iCode1(i+1,n-j) = nc1 - 2*nc2;
     nc1 = nc2;
   endfor
endfor
iCode2 = ones(2**n,n)-iCode1;
toc()

printf("\n");
H = diag(ones(n,1)); c = zeros(n,1);
A1 = []; b1 = []; b2 = [];
tic();
for i=1:mA
   A2 = (ones(2**n,1)*A_low(i,:)).*iCode1 + (ones(2**n,1)*A_up(i,:)).*iCode2;
   A1 = [A1; A2];
   b1 = [b1; b_low(i)*ones(2**n,1)];
   b2 = [b2; b_up(i)*ones(2**n,1)];
endfor
toc()

printf("\n");
A = A1; blow = b1; bup = b2;
n = columns(A),  m = rows(A),
time1 = time() - tstart;


printf("\n"); # set start point and run r-algorithm
x0 = zeros(n,1);
tstart=time();
tic();
[xr,fr,itn,nfg,istop] = ralgb5a(@fgQP,im,x0,alpha,h0,q1,
                                epsg,epsx,maxitn,intp);
printf("..itn %4d  fr %23.15e  istop %d  nfg %4d\n",
       itn, fr, istop, nfg);
toc()
time2 = time() - tstart;

printf("\n");
cxr = 0.5*xr'*H*xr + c'*xr,
fr, #xrt = xr';

printf("\n");
time1, time2,

printf("\n");
n_a_low = n_a_up = 0;
t1 = A*xr; #[blow t1 bup], A,
for i=1:m
   if (abs(t1(i)-blow(i))<0.001) n_a_low = n_a_low + 1; endif
   if (abs(bup(i)-t1(i))<0.001) n_a_up = n_a_up + 1; endif
endfor
n_a_low, n_a_up,
```

Список літератури

1. Nocedal J., Wright S.J. Numerical optimization (Springer Series in Operations Research and Financial Engineering). Berlin: Springer, 2006.
2. Gertz E.M., Wright S.J. Object-oriented software for quadratic programming. ACM Transactions on Mathematical Software (TOMS). 2003. Vol. 29, No. 1. P. 58–81.
3. Boggs P.T., Tolle J.W. Sequential quadratic programming, Acta Numerica. 1995. Vol. 4. P. 1–51.
4. Ben–Tal A., El Ghaoui L., Nemirovski A. Robust Optimization. Princeton: Princeton Univ. Press, 2009. 576 p.
5. Шор Н.З. Методы минимизации недифференцируемых функций и их приложения. Киев: Наук. думка, 1979. 200 с.
6. Octave. http://www.octave.org (звернення: 09.11.2025)
7. Шор Н.З., Стеценко С.И. Квадратичные экстремальные задачи и недифференцируемая оптимизация. Киев: Наук. думка, 1989. 208 с.
8. Стецюк П.И., Фишер А. *r*–Алгоритмы Шора и octave–функция ralgb5a. Тези міжнародної наукової конференції «*Сучасна інформатика: проблеми, досягнення та перспективи розвитку*», що присвячена 60–річчю заснування Інституту кібернетики імені В.М. Глушкова НАН України (Київ, 13–15 грудня 2017). 2017. С. 143–146.
9. 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
10. Bertsekas D.P. Necessary and sufficient conditions for a penalty method to be exact. *Math. Program*. 1975. **9**. P. 87–99. https://doi.org/10.1007/bf01681332
11. Пшеничный Б.Н. Метод линеаризации. М.: Наука, 1983. 136 с.
12. Алєксєєва І.В., Перевознюк Т.І. Застосування робастної оптимізації для лінійної моделі функціонування малого підприємства. *Mathematics in Modern Technical University*. 2018. Vol. 2018, No 1. P. 61–73. https://doi.org/10.20535/mmtu–2018.1–061
13. Stetsyuk P., Fischer A., Pichugina O. A Penalty Approach to Linear Programs with Many Two–Sided Constraints. In: Pardalos P., Khachay M., Kazakov A. (eds) Mathematical Optimization Theory and Operations Research. MOTOR 2021. Lecture Notes in Computer Science. Vol 12755. Springer, Cham. 2021. P. 206–217. https://doi.org/10.1007/978–3–030–77876–7_14
14. Стецюк П.І., Стецюк М.Г., Брагін Д.О., Молодик М.О. Використання r–алгоритму Шора в лінійних задачах робастної оптимізації. *Кібернетика та комп'ютерні технології*. 2021. № 1. С. 29–42. https://doi.org/10.34229/2707–451X.21.1.3
15. Стецюк П., Фішер А., Хом'як О. Штрафна функція максимуму в лінійному програмуванні. *Фізико–математичне моделювання та інформаційні технології*. 2021. № 33. С. 156–160. https://doi.org/10.15407/fmmit2021.33.156
16. Shary S.P. Non–traditional intervals and their use. Which ones really make sense? *Numerical Analysis and Applications*. 2023. **16** (2). P. 179–191. https://doi.org/10.1134/S1995423923020088
17. Gurobi Optimization, Inc., Gurobi Optimizer Reference Manual, 2014. http://www.gurobi.com/ (звернення: 06.08.2025).
18. NEOS Solver. https://neos–server.org/ (звернення: 06.08.2025)
19. Стецюк П.И., Нурминский Е.А. Негладкий штраф и субградиентные алгоритмы для решения задачи проекции на политоп. *Кибернетика и системный анализ*. 2010. № 1. С. 59–63.

Одержано 11.08.2025

Стецюк Петро Іванович,
доктор фізико-математичних наук, завідувач відділу методів негладкої оптимізації
Інституту кібернетики імені В.М. Глушкова НАН України, Київ,
stetsyukp@gmail.com
https://orcid.org/0000–0003–4036–2543

Тукалевська Неля Іванівна,
кандидат фізико-математичних наук, в.о. завідувача відділу методів дискретної оптимізації,
математичного моделювання та аналізу складних систем
Інституту кібернетики імені В.М. Глушкова НАН України, Київ,
Tukalevska@nas.gov.ua

Хом'як Ольга Миколаївна,
кандидат фізико-математичних наук, старший науковий співробітник відділу математичних методів теорії надійності складних систем Інституту кібернетики імені В.М. Глушкова НАН України, Київ,
khomiak.olha@gmail.com
https://orcid.org/0000–0002–5384–9070

Стецюк Марія Григорівна,
інженер-математик відділу методів дискретної оптимізації, математичного моделювання та аналізу складних систем Інституту кібернетики імені В.М. Глушкова НАН України, Київ.
danilyukm5@gmail.com

UDC 519.85

Petro Stetsyuk [*], Nelia Tukalevska, Olha Khomiak, Maria Stetsyuk

R-algorithm for Solving Quadratic Programming Problems

V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv
[*] Correspondence: stetsyukp@gmail.com

Quadratic programming problems have a wide range of practical applications in various fields of science and engineering, particularly in financial modeling and pattern recognition, which underscores the relevance of studying methods for their efficient solution. In real–world applications, the parameters of such models are often imprecise or variable, which can significantly affect the quality of the obtained solutions. Robust approaches ensure the stability of solutions under uncertainty, enabling reliable decisions that remain feasible for all admissible realizations of the input data.

This paper presents the algorithm QPralg (Quadratic Programming by *r*–algorithm) and its implementation in Octave for solving convex quadratic minimization problems with two–sided linear constraints. The proposed method is based on nonsmooth penalty functions and utilizes the Octave program ralgb5a, which implements an *r*-algorithm with adaptive step size. This is a subgradient method with a constant space stretching coefficient in the direction of the difference between two successive subgradients and an adaptive step adjustment along the anti–subgradient direction in the transformed variable space.

The material of the paper is presented in four sections. Section 1 describes the *r*–algorithm with an adaptive step size adjustment mechanism and provides the Octave program ralgb5a that implements it. Section 2 introduces the nonsmooth penalty function method for convex programming problems, along with two theorems that justify the use of finite penalty coefficients. Section 3 describes the QPralg algorithm and its implementation in Octave. Section 4 presents the results of computational experiments for quadratic programming problems with two–sided constraints.

QPralg is particularly suited for problems with a small number of variables and a very large number of constraints (ranging from hundreds of thousands to several millions). For such problems, commercial solvers like Gurobi require several times more computational time on modern computers with 8 GB of RAM. The ability to handle a large number of constraints also enables the effective solution of robust quadratic and linear optimization problems across a wide range of realizations of the uncertainty vector.

Keywords: quadratic programming problem, robust optimization, nonsmooth penalty function, *r*–algorithm, GNU Octave.