КІБЕРНЕТИКА та КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 519.85 DOI:10.34229/2707-451X.21.1.3

П.І. СТЕЦЮК, М.Г. СТЕЦЮК, Д.О. БРАГІН, М.О. МОЛОДИК

ВИКОРИСТАННЯ *r*-АЛГОРИТМУ ШОРА В ЛІНІЙНИХ ЗАДАЧАХ РОБАСТНОЇ ОПТИМІЗАЦІЇ

Робота присвячена опису нового підходу до побудови алгоритмів розв'язання задач лінійного програмування, у яких кількість обмежень є значно більшою за кількість змінних. Він базується на використанні r-алгоритмів для розв`язання задачі мінімізації негладкої функції, яка є еквівалентною задачі лінійного програмування. Переваги підходу продемонстровані на лінійній задачі робастної оптимізації та задачі робастної оцінки параметрів за допомогою методу найменших модулів. Розроблені octave-програми призначені для розв'язання задач лінійного програмування, для яких використання стандартного програмного забезпечення є або неможливим, або недоцільним, адже вимагає значних обчислювальних ресурсів.

Ключові слова: *робастна оптимізація, задача лінійного програмування, негладка штрафна функція, r-алгоритм, метод найменших модулів, GNU Octave.*

Вступ. Задачі робастної оптимізації (задачі з невизначеністю у даних) є звичайними задачами математичного програмування, де цільова функція та обмеження залежать від невизначеної величини [1, 2]. Оптимальний розв'язок робастної задачі є «абсолютно надійним» у тому сенсі, що він вимагає виконання обмежень при умові, що невизначені дані ξ не виходять за межі множини U (множини невизначеності). У статті розглянемо окремий випадок лінійної задачі робастної оптимізації [3], яка має такий вигляд:

$$c^* = \max_{x \in \mathbf{R}^n} c^T x, \tag{1}$$

$$A(\xi)x \le b \ \forall \xi \in U, \tag{2}$$

$$x \ge 0, \tag{3}$$

де матриця обмежень $A(\xi)$ залежить від $\xi \in U \subseteq \mathbf{R}^l$.

Задача (1) – (3) – задача лінійного програмування (ЛП-задача) з нескінченною системою обмежень, якщо множина U є нескінченною. Якщо множина U – скінченна та містить сотні тисяч або мільйони реалізацій вектора ξ, то ЛП-задача (1) – (3) буде характеризуватися невеликою кількістю змінних та дуже великою кількістю обмежень (від сотень тисяч до декількох мільйонів). Для розв'язання таких задач використання стандартного програмного забезпечення з ЛП є або неможливим, або недоцільним, адже вимагає значних обчислювальних ресурсів.

У статті обговоримо спосіб побудови алгоритмів розв'язання ЛП-задач, у яких кількість обмежень значно більша за кількість змінних. Цей спосіб описаний у розділі 2 та базується на негладкій штрафній функції, та виборі параметра штрафу, який забезпечує еквівалентність ЛП-задачі і задачі мінімізації штрафної функції, яка розв'язується за допомогою модифікації *r*-алгоритму Шора [4, 5] (описана у розділі 1). Ця модифікація використана в статті для Octave-реалізації методу найменших модулів, який є робастним до аномальних спостережень або «викидів» (розділ 3).

1. Модифікація r(α)-алгоритму з адаптивним кроком. Розглядається задача пошуку точки мінімуму опуклої функції $f(x)$, $x \in \mathbf{R}^n$. Мінімальне значення функції позначимо $f^* = f(x^*)$, де $x^* \in X^*$. Нехай $\alpha > 1$ – коефіцієнт розтягу простору.

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

$$x_{k+1} = x_k - h_k B_k \xi_k, \quad B_{k+1} = B_k R_\beta(\eta_k), \quad k = 0,1,2,\ldots, \tag{4}$$

де

$$\xi_k = \frac{B_k^T g_f(x_k)}{\left\| B_k^T g_f(x_k) \right\|}, \quad h_k \geq h_k^* = \arg\min_{h \geq 0} f(x_k - h B_k \xi_k), \tag{5}$$

$$\eta_k = \frac{B_k^T r_k}{\| B_k^T r_k \|}, \quad r_k = g_f(x_{k+1}) - g_f(x_k). \tag{6}$$

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 (4) – (5) виконані критерії (умови) зупинки, то вважаємо $k^* = k$, $x_k^* = x_k$ і закінчуємо 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 $A_k = B_k^{-1}$. Дійсно, якщо обидві частини формули $x_{k+1} = x_k - h_k B_k \xi_k$ домножити зліва на матрицю A_k, то отримаємо

$$y_{k+1} = A_k x_{k+1} = A_k x_k - h_k \xi_k = y_k - h_k \frac{B_k^T g_f(x_k)}{\left\| B_k^T g_f(x_k) \right\|} = y_k - h_k \frac{g_\varphi(y_k)}{\left\| g_\varphi(y_k) \right\|}, \tag{7}$$

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 $y = A_k x$.

Сімейство $r(\alpha)$-алгоритмів визначається коефіцієнтом розтягу простору $\alpha > 1$ і послідовністю величин кроків $\{h_k\}_{k=0}^{\infty}$, які визначають ті два послідовні субградієнти $g_f(x_k)$ і $g_f(x_{k+1})$, розтягування за різницею яких (див. формули (6), (7)) зменшує ступінь витягнутості функції у перетвореному просторі змінних. Вибір коефіцієнта $\alpha > 1$ та адаптація величин $\{h_k\}_{k=0}^{\infty}$ до критеріїв зупинки визначають ті чи інші варіанти $r(\alpha)$-алгоритмів. Кількість ітерацій $r(\alpha)$-алгоритму, необхідна для знаходження точки x_{k^*}, для якої $f(x_{k^*}) - f^* \leq \varepsilon$, емпірично оцінюється як $k^* = O(n \log(1/\varepsilon))$, де n – кількість змінних.

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

В практичних реалізаціях $r(\alpha)$-алгоритмів зазвичай використовують адаптивний спосіб регулювання кроку. Він полягає у тому, що величина h_k налаштовується (адаптується) у процесі виконання одновимірного спуску, який завершується, як тільки знайдено субградієнт, що утворює негострий кут з субградієнтом, що визначає напрямок одновимірного спуску (умова завершення спуску за напрямом). Налаштування величини h_k здійснюється за допомогою чотирьох параметрів: $h_0 > 0$ – величина початкового кроку (використовується на першій ітерації, а на кожній наступній – уточнюється), q_1 ($q_1 \le 1$) – коефіцієнт зменшення кроку (якщо умова завершення спуску за напрямком виконується за перший крок), 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 мінімумів X^* – обмежена, то після скінченної кількості кроків адаптивного спуску в напрямку нормованого антисубградієнта обов'язково виконується умова завершення спуску за напрямом.

Для зупинки ітераційного процесу використовуються параметри ε_x і ε_g – алгоритм закінчує роботу в точці $x_{k^*} \in [x_k, x_{k+1}]$, якщо $\|x_{k+1} - x_k\| \le \varepsilon_x$ (зупинка за аргументом), або $\|g_f(x_{k^*})\| \le \varepsilon_g$ (зупинка за нормою градієнта, яка використовується для гладких функцій). Використовуються також стандартна зупинка, якщо перевищено задану максимальну кількість ітерацій **maxitn**, та аварійна зупинка, яка сигналізує про те, що або функція $f(x)$ не є обмеженою знизу, або початковий крок h_0 занадто малий, і його потрібно збільшити.

Програмна реалізація $r(\alpha)$-алгоритму з адаптивним регулюванням кроку за формулами (4) – (6) виконана двома octave-функціями: **ralgb5a** [6] та **ralgb5** [7]. Тут абревіатура "**b5**" означає, що в їх основу покладено r-алгоритм у B-формі, де коректується $n \times n$-матриця B, а кожна ітерація вимагає $5n^2$ арифметичних операцій множення, які визначають обчислювальну трудоємність однієї ітерації. Функція **ralgb5a** – спрощена (для зручності використання) версія **ralgb5**, для якої зафіксовані два найбільш часто вживані параметри $q_2 = 1.1$ і $n_h = 3$. Окрім цього, у функції **ralgb5a** також використовується параметр **intp (int**erval for **p**rint**)**, який забезпечує друк інформації про хід процесу мінімізації через кожні **intp** ітерацій. Цей параметр дозволяє скоротити протокол роботи програми при мінімізації функції для сотень і тисяч змінних, коли 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: $\varepsilon_x \approx 10^{-6}$, $\varepsilon_g \approx 10^{-12}$, **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 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) $g = g_f(x)$ в точці x. Ця функція готується користувачем та може мати довільне ім'я, яке підтримує синтаксис Octave.

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

```
#   Octave-function ralgb5a (Petro Stetsyuk, 07 February 2021)  #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
```

2. *r*(α)-алгоритм для ЛП-задачі ($m \gg n$). Розглянемо ЛП-задачу, яка полягає у максимізації лінійної функції $c(x) = c^T x$ при лінійних нерівностях $Ax \le b$ та $x \ge 0$, де $c \in \mathbf{R}^n$ – вектор коефіцієнтів цільової функції, $x \in \mathbf{R}^n$ – вектор невідомих (змінних), $A = \{a_{ij}\} \in \mathbf{R}^{m \times n}$ – матриця обмежень, $b \in \mathbf{R}^m$ – вектор правих частин. За допомогою цієї ЛП-задачі можна розв'язувати лінійні робастні задачі вигляду (1) – (3), якщо множина U – скінченна та містить сотні тисяч або мільйони реалізацій вектора ξ. Зауважимо, що для розв'язання таких ЛП-задач використання стандартного програмного забезпечення з лінійного програмування є або неможливим або недоцільним, адже вимагає значних обчислювальних ресурсів.

У загальній формі формулювання нашої ЛП-задачі має такий вигляд:

$$\text{знайти } c^* = c(x^*) = \max_{x \in \mathbf{R}^n} \sum_{j=1}^{n} c_j x_j \text{ при обмеженнях } \sum_{j=1}^{n} a_{ij} x_j \le b_i,\ i = 1,\ldots,m,\ x_j \ge 0,\ j = 1,\ldots,n, \quad (8)$$

де c^* – максимальне значення цільової функції, $x^* \in X^* \subset \mathbf{R}^n$ – точка максимуму, X^* – непорожня множина максимумів. Формулювання ЛП-задачі (8) у матричній формі має такий вигляд:

$$\text{знайти } c^* = \max_{x \in \mathbf{R}^n} c^T x \quad \text{при обмеженнях } Ax \le b, \quad x \ge 0. \quad (9)$$

Особливістю ЛП-задач (8) та (9) будемо вважати те, що для них $m \gg n$, тобто кількість лінійних обмежень у цих задачах значно більша за кількість невід'ємних змінних.

За допомогою методу негладких штрафних функцій для задачі (8), а також і для задачі (9), побудуємо еквівалентну допоміжну задачу, яка є задачею безумовної максимізації негладкої увігнутої функції. Для врахування лінійних обмежень $Ax \le b$ та $-x \le 0$ будемо використовувати штрафну функцію у формі функції максимуму.

Нехай $P > 0$ – штрафний коефіцієнт. Розглянемо штрафну функцію

$$c_P(x) = c^T x - P \cdot \max\left\{0, \max_{i=1,\ldots m}\left\{\sum_{j=1}^{n} a_{ij} x_j - b_i\right\}, \max_{j=1,\ldots n}\{-x_j\}\right\} \quad (10)$$

та відповідну їй задачу оптимізації

$$c_P^* = c_P(x_P^*) = \max_{x \in R^n} c_P(x), \quad (11)$$

де c_P^* – максимальне значення функції $c_P(x)$, $x_P^* \in X_P^*$ – точка максимуму, $X_P^* \subset \mathbf{R}^n$ – множина максимумів функції $c_P(x)$.

Задача (11) – задача безумовної максимізації увігнутої кусочно-лінійної функції $c_P(x)$, суперградієнт якої у точці $\bar{x}$ обчислюється за формулою:

$$g_{c_P}(\bar{x}) = c + P \times \begin{cases} 0, & \text{якщо } t_1^* \le 0 \text{ та } t_2^* \le 0, \\ -(a_{i^*1},\ldots,a_{i^*n})^T, & \text{якщо } t_1^* \ge t_2^* \text{ та } t_1^* \ge 0, \\ e_{j^*}, & \text{якщо } t_2^* > t_1^* \text{ та } t_2^* \ge 0, \end{cases} \quad (12)$$

де e_{j^*} – j^*-ий орт, а індекси i^* та j^* визначаються за наступним правилом:

$$i^* :\ t_1^* = \sum_{j=1}^{n} a_{i^* j} \bar{x}_j - b_{i^*} \ge \sum_{j=1}^{n} a_{ij} \bar{x}_j - b_i,\ \forall i = 1,\ldots,m,\ j^* :\ t_2^* = -\bar{x}_{j^*} \ge -\bar{x}_j,\ \forall j = 1,\ldots,n.$$

Розглянемо пару взаємно двоїстих ЛП-задач, де пряма ЛП-задача має вигляд:

$$\text{знайти } \tilde{c}^* = \min_{x\in\mathbf{R}^n} \; -c^T x \quad \text{при обмеженнях } Ax \le b, \quad -x \le 0, \tag{13}$$

а двоїста ЛП-задача має такий вигляд:

$$\text{знайти } \tilde{c}^* = \max_{\Lambda\in\mathbf{R}^m,\Lambda\ge 0} \sum_{j=1}^{m}(-b_i\Lambda_i) \text{ при обмеженнях } \sum_{i=1}^{m} a_{ij}\Lambda_i - \lambda_j = c_j,\ \lambda_j \ge 0,\ j=1,\ldots,n. \tag{14}$$

Зауважимо, що задача (13) має таку ж множину оптимальних розв'язків, як і задачі (8) та (9), а оптимальні значення цільових функцій у цих задачах зв'язані співвідношенням $c^* = -\tilde{c}^*$.

Припустимо, що ЛП-задача (8) має множину оптимальних розв'язків $X^* \subset \mathbf{R}^n$, а для задач (13) та (14) відомі оптимальні множники Лагранжа $\Lambda_1^* \ge 0,\ldots,\Lambda_m^* \ge 0$, які відповідають обмеженням $Ax \le b$, та множники Лагранжа $\lambda_1^* \ge 0,\ldots,\lambda_n^* \ge 0$, які відповідають обмеженням $-x \le 0$.

Теорема 1. Якщо $P \ge P_* = \sum_{i=1}^{m}\Lambda_i^* + \sum_{i=1}^{n}\lambda_i^*$, то $c_P^* = c^*$. Якщо $P > P_*$, то тоді довільна точка $x_P^* \in X^* = X_P^*$, тобто x_P^* – оптимальний розв'язок задачі (8).

Дана теорема – наслідок застосування теореми 27 [8, стор. 23] до ЛП-задачі (13), для якої штрафна функція є опуклою кусково-лінійною функцією та має вигляд

$$\tilde{c}_P(x) = -c^T x + P\cdot\max\left\{0,\ \max_{i=1,\ldots m}\left\{\sum_{j=1}^{n} a_{ij}x_j - b_i\right\},\ \max_{j=1,\ldots n}\left\{-x_j\right\}\right\}, \tag{15}$$

де штрафний коефіцієнт $P > 0$. Враховуючи, що $c_P(x) = -\tilde{c}_P(x)$, отримуємо увігнуту кусково-лінійну штрафну функцію $c_P(x)$ вигляду (10).

Теорема 1 встановлює нижню границю P_* на значення штрафного коефіцієнта, при якому задача (11) – еквівалентна ЛП-задачі (8), а значить еквівалентна і ЛП-задачі (9). Якщо значення штрафного коефіцієнта P вибрати більшим за P_* та розв'язати допоміжну задачу (11), то отримаємо розв'язок задачі (8). Той чи інший метод розв'язання задачі (11), яка є задачею безумовної максимізації негладкої увігнутої функції, забезпечує розв'язання ЛП-задачі (8).

Допоміжну задачу (11) від декількох сотень змінних можна ефективно розв'язувати за допомогою *r*-алгоритму Шора, що означає, що використання Octave-функції **ralgb5a** забезпечить ефективне розв'язання ЛП-задач, у яких кількість обмежень m є значно більшою за кількість змінних $n \approx 100$. Далі покажемо, що за $10-15$ хвилин на сучасних ПЕОМ реалістично розв'язувати ЛП-задачі, в яких щільно заповнена матриця A містить 500 мільйонів елементів та вимагає 4 ГБ оперативної пам'яті для їх зберігання, якщо один елемент матриці займає 8 байт.

Octave-функція **ralgb5a** використовує оракул, який для заданої точки $\bar{x}$ знаходить максимально порушене обмеження ЛП-задачі (8) і визначає значення функції $c_P(\bar{x})$ та її суперградієнта $g_{c_P}(\bar{x})$. Обчислення $c_P(\bar{x})$ за формулою (10) та $g_{c_P}(\bar{x})$ за формулою (12) реалізує octave-функція **fgLP8(**x**)**, для якої n-вимірний вектор x – вектор змінних задачі (8) передається як формальний параметр, а вхідні дані задачі (8) передаються через загальну пам'ять як глобальні змінні. Код octave-функції **fgLP8** є таким:

```
function [f,g] = fgLP8(x)
global c A b n m P
f = sum(c.*x), g = c,
tmp1 = A*x, tmp2 = [tmp1 - b, -x ],
[tmpmax imax] = max(tmp2),
if (tmpmax > 0.d0)
  if (imax <= m)
    f = f - P*tmpmax,
    g = g - P*A(imax,1:n)',
  else
    f = f - P*tmpmax,
    itemp = imax-m,
    g(itemp,1) = g(itemp,1) + P,
  endif
endif
endfunction #fgLP8
```

Вхідні дані задачі (8) для octave-функції **fgLP8** – наступні: **c** – вектор коефіцієнтів цільової функції, **A** – матриця обмежень, **b** – вектор правих частин, **n** – кількість змінних, **m** – кількість обмежень, **P** – штрафний коефіцієнт.

Далі опишемо два обчислювальні експерименти для розв'язання тестових ЛП-задач з великою кількістю обмежень (від двохсот тисяч до п'ятдесяти мільйонів) та невеликою кількістю змінних (від десяти до п'ятдесяти). Умовимось називати їх ЛП-експериментами. Обидва ЛП-експерименти проводились на комп'ютері з процесором Intel Core i5-9400f з тактовою частотою 2,9 ГГц та 16 ГБ оперативної пам'яті.

Мета першого ЛП-експерименту – порівняння результатів роботи *r*-алгоритму з результатами, отриманими за допомогою octave-функції **glpk** [9], яка для розв'язання ЛП-задач використовує відому бібліотеку GLPK [10]. Тут для програми **ralgb5a** використовувалася стартова точка $x_0 = 0$, параметри $r(\alpha)$-алгоритму: $\alpha = 4$, $q_1 = 1.0$, $h_0 = 20.0$ та параметри зупинки: $\varepsilon_x = 10^{-6}$, $\varepsilon_g = 10^{-8}$, maxitn =1500, intp =100. Вхідні дані для ЛП-задач вигляду (8) генерувалися випадковим чином з стандартним рівномірним розподілом $U(0,1)$ за наступними формулами: **c = rand(n,1), A = ones(m,n) + rand(m,n), b = A*ones(n,1)**. ЛП-задачі характеризуються малою кількістю змінних $n \in 10; 20; 50$ та дуже великою кількістю обмежень $m \in 200.000; 500.000; 1.000.000$. Штрафний коефіцієнт P вибирався на одиницю більшим за його нижню межу (див. теорему 1), для чого розв'язувалася ЛП-задача (14) за допомогою octave-функції **glpk**.

Затрати часу **glpk** («t_1») та **ralgb5a** («t_2») на розв'язання задач (8) та (11), абсолютна величина відхилень $dc = \left| c^* - c_P^* \right|$ для отриманих розв'язків («dc»), затрачена **ralgb5a** кількість ітерацій («itn») і обчислень функції та суперградієнта («nfg») при вибраних значеннях штрафного коефіцієнта («P») наведені в табл. 1.

ТАБЛИЦЯ 1. Перший ЛП-експеримент: затрати для функцій glpk (t_1) та ralgb5 (t_2, itn, nfg)

n	m	t_1 (сек)	t_2 (сек)	t_1/t_2	dc	itn	nfg	P
10	200000	1.27	0.83	1.53	1.41e-07	152	282	2.22
	500000	3.13	3.96	0.79	2.32e-07	194	513	2.89
	1000000	6.12	4.29	1.43	7.06e-08	159	273	2.93
20	200000	2.88	2.40	1.20	3.19e-08	387	662	4.45
	500000	7.08	6.60	1.07	9.26e-08	393	709	4.61
	1000000	14.41	10.34	1.39	4.37e-08	343	553	4.63
50	200000	9.77	17.10	0.57	2.94e-08	1946	3120	15.06
	500000	25.87	32.48	0.80	3.04e-08	1553	2383	13.25
	1000000	52.40	59.45	0.88	9.66e-08	1411	2155	13.80

З даної таблиці видно, що для розв'язання відповідних ЛП-задач за допомогою **glpk** вимагається приблизно стільки ж часу, як і для розв'язання допоміжної задачі (11) за допомогою **ralgb5a.** Результати колонки «dc» показують, що відхилення між розв'язками ЛП-задачі та допоміжної задачі є дуже малими, що означає, що обидві програми збігаються до одного і того ж єдиного розв'язку для кожної з тестових ЛП-задач.

Мета другого ЛП-експерименту – оцінка часу програми **ralgb5a** для розв'язання таких же ЛП-задач, як і в першому експерименті, де матриця A містить 500 мільйонів елементів. Враховуючи, що для зберігання такої матриці потрібно 4 ГБ оперативної пам'яті (вважається, що один елемент матриці займає 8 байт), то цю ЛП-задачу умовимось називати «**4ГБ**-задачею». Параметри $r(\alpha)$-алгоритма для другого ЛП-експерименту вибиралися такими ж як і для першого, за винятком $\alpha = 3$. Штрафний коефіцієнт P вибирався рівним 50. Другий експеримент реалізує наведений далі Octave -код.

```
#  Code for time of ralgb5a for 4Gb-problems
global c A b n m P
printf("\n"), # set parameters for r-algorithm
alpha = 3.0, h0 = 20.0, q1 = 1.0,
epsx = 1.e-6, epsg = 1.e-8, maxitn = 1500, intp = 200,
nmtest = [ 50 10000000,
           20 25000000,
           10 50000000],
P = 50, rand("seed", 2020), time12 = fopt = itna = [],
for itest =  1:rows(nmtest)
   printf("\n"), # set test example
   n = nmtest(itest,1), m = nmtest(itest,2),
   c = rand(n,1), A = ones(m,n)+rand(m,n),
   x0 = ones(n,1), b = A*x0,
   # set start point and run r-algorithm
   im = -1, x0 = zeros(n,1), tstart = time(),
   [xr,fr,itn,nfg,ist]=ralgb5a(@fgLP8,im,x0,alpha,h0,q1,
                               epsg,epsx,maxitn,intp),
   time1=time()-tstart,
   printf("..itn %4d  fr %23.15e  ist %d  nfg %4d\n",itn,fr,ist,nfg),
   fr, ctxr = c'*xr, time1, time12 = [time12, time1],
   fopt = [ fopt, c'*xr fr], itna = [itna, itn nfg ist],
endfor
nmtest, fopt, # print of results
printf("\n   n      m      .t1..  ist .itn. .nfg."),
for itest =  1:rows(nmtest)
  n = nmtest(itest,1), m = nmtest(itest,2),
  t1 = time12(itest,1),
  printf("\n  %3d  %6d  %7.2f",n,m,t1),
  itn = itna(itest,1), nfg = itna(itest,2),
  ist = itna(itest,3),
  printf(" %2d  %5d %5d",ist,itn,nfg),
endfor
printf("\n"),
```

Результати розрахунків для $n=50,20,10$ та $m=10.000.000,25.000.000,50.000.000$ наведено в табл. 2, де ist – код завершення роботи програми **ralgb5a.**

ТАБЛИЦЯ 2. Другий ЛП-експеримент: затрати **ralgb5a** для розв'язання **4ГБ**-задач

n	m	$time$ (сек.)	itn	nfg	ist
50	10.000.000	625.75	1500	2310	4
20	25.000.000	479.05	575	1028	3
10	50.000.000	858.45	368	1104	3

З даної таблиці (колонка «$time$») видно, що за 10 – 15 хвилин на сучасних ПЕОМ реалістично розв'язувати ЛП-задачі з $n=10\div 50$, в яких щільно заповнена матриця A містить 500 мільйонів елементів та вимагає 4 ГБ оперативної пам'яті для їх зберігання. Зауважимо, що Octave-реалізація **ralgb5a** не претендує на ефективну за часом розв'язання задачі (11). Для неї витрати за часом можна зменшити в десятки разів, якщо програму переписати мовою python, для якої бібліотека NumPy працює значно швидше і реалізована на C і Fortran (переважно матричні операції) й спирається на коди бібліотек BLAS, ATLAS та LAPACK.

3. Метод найменших модулів ($m\gg n$). Нехай для оцінки n невідомих параметрів $x_1,..,x_n$ використовується m спостережень $y_1,..,y_m$ причому ці величини пов'язані співвідношенням:

$$y_i=\sum_{j=1}^{n}a_{ij}x_j+u_i,\quad i=1,...,m, \tag{16}$$

де a_{ij} – відомі коефіцієнти, u_i – невідомі випадкові величини, що мають (приблизно) однакові функції розподілу. Рівняння (16) можна записати в матричній формі:

$$y=Ax+u, \tag{17}$$

де $y=\left(y_1,..,y_m\right)^T\in\mathbf{R}^m$ і $u=\left(u_1,..,u_m\right)^T\in\mathbf{R}^m$ – m-вимірні вектори, A – матриця розміру $m\times n$, $x=\left(x_1,..,x_n\right)^T\in\mathbf{R}^n$ – m-вимірний вектор параметрів, які потрібно оцінити.

У класичній постановці метод найменших модулів (відповідає знаходженню невідомого вектора x^* згідно з критерієм найменших модулів) – задача математичного програмування:

$$f^*_{LMM}=f_{LMM}\left(x^*\right)=\min_{x\in\mathbf{R}^n}\left\{f_{LMM}\left(x\right)=\sum_{i=1}^{m}\left|y_i-\sum_{j=1}^{n}a_{ij}x_j\right|\right\}, \tag{18}$$

де $|\cdot|$ – модуль (абсолютна величина) числа. Задача (18) – безумовна задача мінімізації опуклої кусково-лінійної функції. Зауважимо, що метод найменших модулів є робастним до аномальних спостережень або «викидів» [11, 12].

Задача (18) – безумовна задача мінімізації опуклої кусково-лінійної функції $f_{LMM}\left(x\right)$, субградієнт якої у точці $\overline{x}$ обчислюється за такою формулою:

$$g_{f_{LMM}}(\overline{x})=\left(\sum_{i=1}^{m}sign\left(\sum_{j=1}^{n}a_{ij}\overline{x}_j-y_i\right)a_{i1},\sum_{i=1}^{m}sign\left(\sum_{j=1}^{n}a_{ij}\overline{x}_j-y_i\right)a_{i2},\ldots,\sum_{i=1}^{m}sign\left(\sum_{j=1}^{n}a_{ij}\overline{x}_j-y_i\right)a_{in}\right)^T. \tag{19}$$

Знаходження найкращого за критерієм найменших модулів вектора x^*, де x^* – розв'язок задачі (18), можна звести до розв'язання наступної ЛП-задачі:

$$\text{знайти } f^*_{LMM} = \min_{z\in\mathbf{R}^n, z\geq 0} \sum_{i=1}^{m} z_i \text{ при обмеженнях } y_i - \sum_{j=1}^{n} a_{ij}x_j \leq z_i, -y_i + \sum_{j=1}^{n} a_{ij}x_j \leq z_i, i = 1,\ldots,m. \quad (20)$$

Для розв'язання ЛП-задачі (20) можна використовувати відповідні стандартні програми лінійного програмування. При цьому паралельно зі знаходженням самого вектора x^* знаходяться і оптимальні значення вектора $z^* = \left(z_1^*,..,z_m^*\right)^T$, компоненти якого задають оцінки для незалежної випадкової величини $u_i, i = 1,..,m.$

Далі наведемо результати двох обчислювальних експериментів для розв'язання тестових задач (18) з невеликою кількістю невідомих параметрів (від десяти до ста), де кількість спостережень $m \gg n$. Будемо їх називати МНМ-експериментами. Обидва МНМ-експерименти проводились на тому ж комп'ютері, що і ЛП-експерименти із розділу 2.

Мета першого МНМ-експерименту – порівняння результатів роботи програми **ralgb5a** з результатами, отриманими за допомогою octave-функції **glpk** для розв'язання ЛП-задач (20). Для програми **ralgb5a** використовувалася стартова точка $x_0 = 0$, параметри $r(\alpha)$-алгоритму: $\alpha = 3$, $q_1 = 0.95$, $h_0 = 5.0$ та параметри зупинки: $\varepsilon_x = 10^{-8}$, $\varepsilon_g = 10^{-8}$, maxitn=1500, intp=100. Вхідні дані для задачі (18) генерувалися випадковим чином з стандартним рівномірним розподілом $U(0,1)$ за наступними формулами: **A = rand(m,n), b = A*ones(n,1), b(m,1) = b(m,1) + 1**.

Для заданої точки $\bar{x}$ обчислення $f_{LMM}(\bar{x})$ та її субградієнта $g_{f_{LMM}}(\bar{x})$ за формулою (19) реалізує octave-функція **fgLMM(**x**)**, для якої n-вимірний вектор x передається як формальний параметр, а вхідні дані задачі (18) передаються через загальну пам'ять як глобальні змінні. Код octave-функції **fgLMM** є таким:

```
function [f,g] = fgLMM(x)
global A y n m
tmp = A*x - y,
f = sum(abs(tmp)),
A1 = A.*(sign(tmp)*ones(1,n)),
g1 = sum(A1,1),
g = g1',
endfunction #fgLMM
```

Вхідні дані задачі (18) для octave-функції **fgLMM** є наступними: **A** – матриця відомих коефіцієнтів, **y** – вектор результатів спостережень, **n** – кількість параметрів, **m** – кількість спостережень.

Результати першого МНМ-експерименту – затрати часу **glpk** («t_1», «t_2») на розв'язання задач (19) відповідно прямим та двоїстим симплекс-методами, затрати **ralgb5a** («t_3», «itn», «nfg») на розв'язання задач (18), $\left\|x_r - x^*\right\|$ – норма відхилення знайденого за допомогою **ralgb5a** наближення до точки мінімуму від відомої точки мінімуму $x^* = (1,1,\ldots,)^T$ наведені в табл. 3.

ТАБЛИЦЯ 3. Перший МНМ-експеримент: затрати для функцій glpk (t_1, t_2) та ralgb5 (t_3, *itn*, *nfg*)

n	*m*	t_1 (сек)	t_2 (сек)	t_3 (сек)	$\|x_r - x^*\|$	*itn*	*nfg*
100	20000	78.65	142.65	10.59	7.59e-09	481	651
	10000	16.09	22.29	5.33	6.36e-09	449	584
50	20000	51.81	92.47	4.18	3.26e-09	382	508
	10000	5.59	10.40	1.86	4.92e-09	358	456
20	20000	52.89	101.91	1.22	5.72e-09	236	320
	10000	4.28	8.54	0.68	2.48e-09	230	305
10	20000	33.87	74.28	0.63	5.82e-09	149	214
	10000	4.07	8.17	0.12	5.44e-09	142	188

З даної таблиці видно, що для розв'язання відповідних ЛП-задач за допомогою **glpk** вимагається в десятки разів більше часу, ніж для розв'язання задачі (18) за допомогою **ralgb5a.** Результати колонки «$\|x_r - x^*\|$» показують, що відхилення між знайденим розв'язком задачі (18) та її точкою мінімуму є дуже малим та відповідає параметру $\varepsilon_x = 10^{-8}$ для зупинки *r*-алгоритма.

Мета другого обчислювального МНМ-експерименту – оцінка часу програми **ralgb5a** для розв'язання задачі (18) з малою кількістю змінних $n \in 10; 20; 50; 80; 100$, де матриця A містить 50 мільйонів елементів, тобто в 10 раз менше, ніж це мало місце для другого ЛП-експерименту. Враховуючи, що для зберігання такої матриці потрібно 400 МБ оперативної пам'яті, то такі задачі для методу найменших модулів будемо називати «**4ГБ**-задачами». Вхідні дані для задачі (18) генерувалися за формулами: **A = rand(m,n), b = A*ones(n,1)**. Параметри $r(\alpha)$ -алгоритма для другого МНМ-експерименту вибиралися такими ж як і для першого, за винятком $q_1 = 1.0$ та $\varepsilon_x = 10^{-6}$. Четвертий експеримент реалізує наведений нижче Octave -код.

```
#  Code for time of ralgb5a for 400Mb-problems
global A y n m
printf("\n"), # set parameters for r-algorithm
alpha = 3.0, h0 = 5.0, q1 = 1.0,
epsx = 1.e-6, epsg = 1.e-8, maxitn = 1500, intp = 100,
nmtest = [ 100 500000, 80 625000, 50 1000000, 20 2500000, 10 5000000],
rand("seed", 2020), time12 = fopt = itna = [],
for itest =  1:rows(nmtest)
   printf("\n"), # set test example
   n = nmtest(itest,1), m = nmtest(itest,2),
   A = ones(m,n)+rand(m,n), x0 = ones(n,1), y = A*x0,
   # set start point and run r-algorithm
   im = 1, x0 = zeros(n,1), tstart = time(),
   [xr,fr,itn,nfg,ist]=ralgb5a(@fgLMM,im,x0,alpha,h0,q1,
                               epsg,epsx,maxitn,intp),
   time1=time()-tstart,
   printf("..itn %4d  fr %23.15e  ist %d  nfg %4d\n",itn,fr,ist,nfg),
   fr, time1, time12 = [time12, time1],
   fopt = [ fopt, fr norm(xr-ones(n,1))], itna = [itna, itn nfg ist],
endfor
nmtest, fopt,
printf("\n   n       m       .t1..    dx      ist .itn. .nfg."),
for itest =  1:rows(nmtest)
  n = nmtest(itest,1), m = nmtest(itest,2),
  t1 = time12(itest,1), fr = fopt(itest,2),
  printf("\n  %3d  %8d  %7.2f %9.2e ",n,m,t1,fr),
  itn = itna(itest,1), nfg = itna(itest,2), ist = itna(itest,3),
  printf(" %2d  %5d %5d",ist,itn,nfg),
endfor
printf("\n"),
```

Результати розрахунків для $n = 10 \div 100$ та $m = 500.000 \div 5.000.000$ наведено в табл. 4, де *ist* – код завершення роботи програми **ralgb5a,** $\|x_r - x^*\|$ – норма відхилення знайденого наближення до точки мінімуму від самої точки мінімуму $x^* = (1,1,\ldots,)^T$.

ТАБЛИЦЯ 4. Другий МНМ-експеримент: затрати **ralgb5a** для розв'язання **400МБ**-задач

n	*m*	*time* (сек.)	*itn*	*nfg*	*ist*	$\|x_r - x^*\|$
100	500.000	754.45	1500	1831	4	3.18e-06
80	625.000	689.47	1343	1670	3	6.26e-07
50	1.000.000	441.08	829	1055	3	3.68e-07
20	2.500.000	178.07	312	409	3	5.66e-07
10	5.000.000	96.01	147	198	3	6.60e-07

З колонки «*time* » даної таблиці видно, що за допомогою сучасних ПЕОМ за десять-двадцять хвилин можна розв'язувати оптимізаційні задачі для методу найменших модулів з невеликою кількістю параметрів ($n = 10 \div 100$), якщо матриця A вимагає 400 МБ оперативної пам'яті для зберігання коефіцієнтів. Зауважимо, що такі розміри матриці A не дозволяють використати стандартне програмне забезпечення з ЛП, так як воно вимагає значних обчислювальних ресурсів із-за збільшення кількості змінних в ЛП-задачах вигляду (19). За аналогічною схемою, як і для ЛП-задач у розділі 2, для методу найменших модулів витрати за часом можна значно зменшити, якщо Octave-програми **ralgb5a** та **fgLMM** переписати мовою python з використанням бібліотеки NumPy.

Затрати програми **ralgb5a** на розв'язання першої та другої задач з табл. 4 можна суттєво зменшити за рахунок регулювання параметрів *r*-алгоритму. Так наприклад, якщо замість $q_1 = 1.0$ використати $q_1 = 0.95$, то для задачі з $n = 100$ та $m = 500.000$ затрати програми **ralgb5a** складають $time = 239.71$, $itn = 411$ та $nfg = 578$, а для задачі з $n = 80$ та $m = 625.000$ вони складають $time = 258.09$, $itn = 422$ та $nfg = 626$.

Висновки. У статті описано новий підхід до побудови алгоритмів розв'язання задач лінійного програмування, у яких кількість обмежень є значно більшою за кількість змінних. Він заснований на використанні *r*-алгоритмів для розв`язання безумовної задачі мінімізації негладкої штрафної функції, яка є еквівалентною задачі лінійного програмування. Переваги запропонованого підходу продемонстровані на лінійній задачі робастної оптимізації та задачі знаходження робастної оцінки параметрів за допомогою методу найменших модулів.

Розроблені octave-програми призначені для розв'язання задач лінійного програмування з дуже великою кількістю обмежень, для яких використання стандартного програмного забезпечення є або неможливим або недоцільним, адже вимагає значних обчислювальних ресурсів. Це дозволить забезпечити розв`язання на сучасних ПЕОМ лінійних задач робастної оптимізації вигляду (1) – (3), у яких множина U – скінченна та містить сотні тисяч або мільйони реалізацій вектора ξ .

Моделі робастних ЛП-задач вигляду (1) – (3) мають місце в економічних застосуваннях. В роботі [3] розглядається задача оптимізації доходу малого підприємства, де цільова функція відображає максимальний дохід підприємства при ресурсних обмеженнях, або інших технологічних обмеженнях (наприклад, кількість працівників, кількість робочих годин і т. д). Робастний підхід дає можливість прогнозування розвитку підприємства навіть тоді, коли вхідні параметри мають невизначені збурення (в роботі [3] збурення сягали 20 % від своїх номінальних значень).

В книзі [1] описано приклад максимізації прибутку підприємства, яке виготовляє ліки. На виготовлення партії ліків можна використовувати два види сировини, які мають різну кількість активної речовини і різну вартість. Для виконання плану (визначається кількість упаковок, щоб укомплектувати партію) при застосуванні лінійної оптимізації перевага буде повністю віддана «дешевшій» сировині. Проте, якщо активна речовина видобувається з деяким збуренням з кожного екземпляру сировини (в задачі розглядаються збурення 0,5 % для першого виду і 2 % для другого), то за стратегією лінійної оптимізації, у найгіршому випадку можна отримати не тільки збитки, але й недоукомплектувати партію. Тому при робастному підході буде надаватися перевага «надійності розв'язку» і максимізації доходу і розв'язок може включати не тільки дешевшу «ризиковану» сировину, але дорожчу більш «надійну».

Список літератури

1. Ben-Tal A., El Ghaoui L., Nemirovski A. Robust Optimization. Princeton: Princeton Univ. Press, 2009. 576 p.
2. Горяшко А.П., Немировский А.С. Оптимизация стоимости энергии в системах водоснабжения в условиях неопределенности потребления. *Автоматика и телемеханика*. 2014. Выпуск 10. С. 52–72.
3. Алєксєєва І.В., Перевознюк Т.І. Застосування робастної оптимізації для лінійної моделі функціонування малого підприємства. *Mathematics in Modern Technical University*. 2018. 1. P. 61–73. https://doi.org/10.20535/mmtu-2018.1-061
4. Шор Н.З. Методы минимизации недифференцируемых функций и их приложения. Киев: Наукова думка, 1979. 200 с.
5. Стецюк П.И. Теория и программные реализации *r*-алгоритмов Шора. *Кибернетика и системный анализ*. 2017. № 5. С. 43–57.
6. 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). К.: Ін-т кібернетики імені В.М. Глушкова НАН України. 2017. С. 143–146.
7. Стецюк П.И. Субградиентные методы ralgb5 и ralgb4 для минимизации овражных выпуклых функций. *Вычислительные технологии*. 2017. Т. 22, № 2. С. 127–149.
8. Shor N.Z. Non-Differentiable Optimization and Polynomial Problems. Kluwer Academic Publishers, Boston, Dordrecht, London. 1998. 412 p.
9. Eaton J.W., Bateman D., Hauberg S. GNU Octave Manual Version 3. Network Theory Ltd. 2008. 568 p.
10. GNU Linear Programming Kit (GLPK). http://www.gnu.org/software/glpk/glpk.html (звернення 12.03.2021)
11. Хьюбер Дж. П. Робастность в статистике. М.: Мир, 1984. 304 с.
12. Стецюк П.И., Колесник Ю.С., Лейбович М.М. О робастности метода наименьших модулей. *Компьютерная математика*. 2002. С. 114–123.

Одержано 12.03.2021

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

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

Брагін Даніїл Олександрович,
студент Московського фізико-технічного інституту, Долгопрудний, Московська область,
bragin.da@phystech.edu

Молодик Микола Олександрович,
студент Московського фізико-технічного інституту, Долгопрудний, Московська область.
molodyk.na@phystech.edu

UDC 519.85

P.I. Stetsyuk [1 *], M.G. Stetsyuk [1], D.A. Bragin [2], N.A. Molodyk [2]

Use of the Shor's *r*-Algorithm in Linear Robust Optimization Problems

[1] *V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv*
[2] *Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region*
[*] *Correspondence: stetsyukp@gmail.com*

The paper is devoted to the description of a new approach to the construction of algorithms for solving linear programming problems (LP-problems), in which the number of constraints is much greater than the number of variables. It is based on the use of a modification of the *r*-algorithm to solve the problem of minimizing a nonsmooth function, which is equivalent to LP problem. The advantages of the approach are demonstrated on the linear robust optimization problem and the robust parameters estimation problem using the least moduli method. The developed octave programs are designed to solve LP problems with a very large number of constraints, for which the use of standard software from linear programming is either impossible or impractical, because it requires significant computing resources.

The material of the paper is presented in three sections. In the first section for the problem of minimizing a convex function we describe a modification of the *r*-algorithm with a constant coefficient of space dilation in the direction of the difference of two successive subgradients and an adaptive method for step size adjustment in the direction of the antisubgradient in the transformed space of variables. The software implementation of this modification is presented in the form of Octave function ralgb5a, which allows to find or approximation of the minimum point of a convex function, or approximation of the maximum point of the concave function. The code of the ralgb5a function is given with a brief description of its input and output parameters.

In the second section, a method for solving the LP problem is presented using a nonsmooth penalty function in the form of maximum function and the construction of an auxiliary problem of unconstrained minimization of a convex piecewise linear function. The choice of the finite penalty coefficient ensures equivalence between the LP-problem and the auxiliary problem, and the latter is solved using the ralgb5a program. The results of computational experiments in GNU Octave for solving test LP-problems with the number of constraints from two hundred thousand to fifty million and the number of variables from ten to fifty are presented.

The third section presents least moduli method that is robust to abnormal observations or "outliers". The method uses the problem of unconstrained minimization of a convex piecewise linear function, and is solved using the ralgb5a program. The results of computational experiments in GNU Octave for solving test problems with a large number of observations (from two hundred thousand to five million) and a small number of unknown parameters (from ten to one hundred) are presented. They demonstrate the superiority of the developed programs over well-known linear programming software such as the GLPK package.

Keywords: robust optimization, linear programming problem, nonsmooth penalty function, *r*-algorithm, least modulus method, GNU Octave.