КІБЕРНЕТИКА та КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 519.85 DOI:10.34229/2707-451X.22.3.10

П.І. СТЕЦЮК, О.В. ПИЛИПОВСЬКИЙ, О.М. ХОМ'ЯК

GNU OCTAVE ТА PYTHON РЕАЛІЗАЦІЇ r-АЛГОРИТМУ ШОРА З АДАПТИВНИМ РЕГУЛЮВАННЯМ КРОКУ

Робота присвячена опису двох програмних реалізацій r-алгоритму Шора з адаптивним регулюванням кроку мовою GNU Octave та мовою Python. Описано тестові експерименти для дослідження ефективності octave-функції ***ralgb5a*** *для максимізації кусково-лінійної увігнутої функції (метод негладких штрафних функцій для спеціальної задачі лінійного програмування) та мінімізації кусково-лінійної опуклої функції (метод найменших модулів). Описані результати обчислювальних експериментів для python-функції* ***ralgb5a.***

Ключові слова: *r-алгоритм, лінійне програмування, негладка штрафна функція, метод найменших модулів, GNU Octave, python.*

Вступ. Субградієнтні методи, в яких використовується розтяг простору в напрямку різниці двох послідовних субградієнтів, запропоновані в [1, 2]. Вони отримали назву *r*-алгоритмів і стали одним із центральних результатів докторської дисертації Н.З. Шора (1970). Програмні реалізації *r*-алгоритмів виявилися конкурентними як за надійністю, так і за часом розрахунків та точності результатів з найбільш ефективними методами для гладких погано обумовлених задач. Прискорену збіжність *r*-алгоритмів при мінімізації (максимізації) негладких опуклих (увігнутих) функцій забезпечує поєднання у них двох пов'язаних ідей.

Перша ідея – використання процедури найшвидшого спуску у напрямку антисубградієнта опуклої функції у перетвореному просторі змінних. Вона гарантує монотонність за значеннями опуклої функції для точок мінімізуючої послідовності, яка конструюється *r*-алгоритмами. Якщо пошук мінімуму функції у напрямку антисубградієнта здійснюється приблизно, тоді "монотонність" за мінімізованою функцією замінюється на "майже монотонність". Друга ідея – використання операції розтягування простору у напрямку різниці двох послідовних субградієнтів, де другий субградієнт обчислений у точці мінімуму функції у напрямку першого антисубградієнта. Внаслідок цього розтягування зменшуються поперечні складові субградієнтів уздовж напрямку на точку мінімуму, що дозволяє зменшити ступінь "витягнутості" яружної функції у перетвореному просторі змінних.

У статті описано програмні реалізації *r*-алгоритму Шора з адаптивним регулюванням кроку [3 – 5] мовою GNU Octave [6] (розділ 1) та мовою Python (розділ 3). У розділі 2 описано тестові експерименти для дослідження ефективності octave-функції **ralgb5a** [7] для максимізації кусково-лінійної увігнутої функції (метод негладких штрафних функцій для спеціальної задачі лінійного програмування) та мінімізації кусково-лінійної опуклої функції (метод найменших модулів). В розділі 3 описані результати обчислювальних експериментів для python-функції **ralgb5a.**

1. Octave-функція ralgb5a. Реалізує модифікацію *r(α)*-алгоритму з адаптивним регулюванням величини кроку для 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.

$r(\alpha)$-алгоритмом мінімізації функції $f(x)$ називається ітеративна процедура знаходження послідовностей n-вимірних векторів $\{x_k\}_{k=0}^{\infty}$ та $n \times n$-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 $f(x)$ у напрямку $-B_k \xi_k$, $R_\beta(\eta) = I_n + (\beta - 1)\eta\eta^T$ – 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 критерії (умови) зупинки, то вважаємо $k^* = k$, $x_k^* = x_k$ і закінчуємо роботу алгоритму.

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

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

де вектор $g_\varphi(y_k) = B_k^T g_f(x_k)$ – субградієнт 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 (3), (4)) зменшує ступінь витягнутості функції у перетвореному просторі змінних. Вибір коефіцієнта $\alpha > 1$ та адаптація величин $\{h_k\}_{k=0}^{\infty}$ до субградієнтного спуску та критеріїв зупинки визначають ті чи інші варіанти $r(\alpha)$-алгоритмів. При правильному виборі параметрів 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.

Адаптивний спосіб регулювання кроку в напрямі антисубградієнта в перетвореному просторі змінних 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 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 octave-функцією **ralgb5** [8]. Тут абревіатура "**b5**" означає, що коректується $n \times n$-матриця B, а кожна ітерація вимагає $5n^2$ арифметичних операцій множення, які визначають обчислювальну складність однієї ітерації. Програма **ralgb5** використовує octave-функцію **function [f, g] = = calcfg (x)**, яка обчислює значення функції $f = f(x)$ та її субградієнта $g = g_f(x)$ у точці x. Ця функція готується користувачем та може мати довільне ім'я, яке підтримує синтаксис Octave.

Спрощена (для зручності використання) версія **ralgb5**, для якої зафіксовані два найбільш часто вживані параметри $q_2 = 1.1$ і $n_h = 3$, реалізована octave-функцією **ralgb5a** [9]. Окрім цього, у функції **ralgb5a** 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.

У роботі [10] octave-функція **ralgb5a** доповнена параметром **im**: якщо **im=1**, то x_r^* – наближення до точки мінімуму опуклої функції, якщо **im=-1**, то x_r^* – наближення до точки максимуму увігнутої функції. При цьому **ralgb5** використовує octave-функцію **calcfg**, яка обчислює значення опуклої (увігнутої) функції $f = f(x)$ та її субградієнта (суперградієнта) $g = g_f(x)$ у точці x.

Якщо ітераційний процес запускається з 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: $\varepsilon_x \approx 10^{-6}$, $\|x_r - x^*\|$, **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.

Код 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. Два обчислювальних експерименти. Наведемо результати оптимізації для двох негладких функцій з кількістю змінних $n = 200 \div 2000$ за допомогою octave-функції **ralgb5a**. Обчислення проводились на комп'ютері з процесором Intel Core i5-9400f з тактовою частотою 2,9 ГГц та 16 ГБ оперативної пам'яті з використанням GNU Octave версії 5.1.0. Перший експеримент назвемо ЛП-експериментом, так як він пов'язаний зі знаходженням розв'язку задачі лінійного програмування (ЛП-задачі) такого вигляду:

$$\text{знайти } c^* = \max_{x \in \mathbf{R}^n} \left\{ c^T x \right\} \text{ при обмеженнях } Ax \le b, \quad x \ge 0, \tag{5}$$

де $c \in \mathbf{R}^n$ – вектор коефіцієнтів цільової функції, $x \in \mathbf{R}^n$ – вектор невідомих (змінних), $A = \left\{ a_{ij} \right\} \in \mathbf{R}^{m \times n}$ – матриця обмежень з компонентами a_{ij}, $b \in \mathbf{R}^m$ – вектор правих частин з додатними компонентами b_i, $c^* > 0$ – максимальне значення цільової функції, $x^* \in X^* \subset \mathbf{R}^n$ – точка максимуму, X^* – непорожня множина максимумів.

За допомогою методу негладких штрафних функцій задачу (5) можна звести до еквівалентної задачі безумовної максимізації негладкої увігнутої функції такого вигляду:

$$c_P^* = c_P(x_P^*) = \max_{x \in R^n} \left\{ c_P(x) = c^T x - P \cdot \max \left\{ 0, \max_{i=1,\dots m} \left\{ \frac{1}{b_i} \left(\sum_{j=1}^{n} a_{ij} x_j - b_i \right) \right\}, \max_{j=1,\dots n} \left\{ -x_j \right\} \right\} \right\}, \tag{6}$$

де c_P^* – максимальне значення функції $c_P(x)$, x_P^* – точка максимуму функції $c_P(x)$. Якщо значення штрафного коефіцієнта P вибрати більшим за c^* та розв'язати задачу (6), то $x_P^* \in X^*$ і ми отримаємо розв'язок ЛП-задачі (5).

Задача (6) – задача безумовної максимізації увігнутої кусково-лінійної функції $c_P(x)$, значення якої **f** та суперградієнт **g** обчислюється за формулами octave-функції **fgLP6**:

```
function [f,g] = fgLP6(x)
global c A b n m P # parameters for problem (6)
f = sum(c.*x); g = c;
tmp1 = A*x - b; tmp2 = [tmp1./b; -x];
[tmpmax imax] = max(tmp2);
if (tmpmax > 0.d0)
  if (imax <= m)
    f = f - P*tmpmax;
    g = g - (P/b(imax))*A(imax,1:n)';
  else
    f = f - P*tmpmax;
    itemp = imax - m;
    g(itemp,1) = g(itemp,1) + P;
  endif
endif
endfunction #fgLP6
```

Вхідні дані задачі (6) для octave-функції **fgLP6** – наступні: **c** – вектор коефіцієнтів цільової функції, **A** – матриця обмежень, **b** – вектор правих частин, **n** – кількість змінних, **m** – кількість обмежень, **P** – штрафний коефіцієнт. Вони передаються функції **fgLP6** через глобальні змінні.

Мета ЛП-експерименту – оцінка часу, необхідного програмі **ralgb5a** для розв'язання задачі (6) при $n \in \{200, 500, 1000, 1500, 2000\}$ та $m = 2n$. Вхідні дані генерувалися випадковим чином з стандартним рівномірним розподілом $U(0,1)$ за наступними формулами: **c = 0.1*rand(n,1), A = ones(m,n) + rand(m,n), b = A*ones(n,1)**. Використовувалася стартова точка $x_0 = 0$ та параметри *nfg* -алгоритму: $\alpha = 2$, $q_1 = 0.9$, $h_0 = 20.0$ та параметри зупинки: $\varepsilon_x = 10^{-3}$, $\varepsilon_g = 10^{-8}$, maxitn =10000, intp =1000. Штрафний коефіцієнт P вибирався рівним 1000. ЛП-експеримент реалізує наведений далі octave -код.

```
# computational efficiency of ralgb5a for problem (6)
global c A b n m P
printf("\n"), # set parameters for r-algorithm
alpha = 2.0, h0 = 20.0, q1 = 0.9,
epsx = 1.e-3, epsg = 1.e-8, maxitn = 10000, intp = 1000,
nmtest = [ 200 400; 500 1000; 1000 2000; 1500 3000; 2000 4000];
P = 1000, rand("seed", 2022); time12 = fopt = itna = [];
for itest = 1:rows(nmtest)
   printf("\n"); # set test example
   n = nmtest(itest,1); m = nmtest(itest,2);
   c = 0.1*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(@fgLP6,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 наведено в табл. 1, де f_r – значення функції, що максимізується, *time* – час (в секундах), затрачений на розв'язання задачі (6), *itn* – кількість ітерацій, *nfg* – кількість викликів функції, *ist* – код завершення роботи програми **ralgb5a.**

ТАБЛИЦЯ 1. ЛП-експеримент: затрати octave-функції **ralgb5a** для розв'язання тестових задач (6)

n	*m*	f_r	*time*	*itn*	*nfg*	*ist*
200	400	16.030	0.80	1035	3430	3
500	1000	25.607	1.37	502	1500	3
1000	2000	59.953	7.55	811	2696	3
1500	3000	89.703	16.14	739	2460	3
2000	4000	122.82	23.88	605	1985	3

З табл. 1 (колонка «*time*») видно, що за 10 – 30 секунд на сучасних ПЕОМ octave-реалізація **ralgb5a** дозволяє розв'язувати тестові ЛП-задачі з $n=1000\div2000$, $m=2000\div4000$. Так, наприклад, для тестового прикладу при $n=1500$, $m=3000$ затрачено 20 с, а при $n=2000$, $m=4000$ – близько 30 с. Зауважимо, що на одну ітерацію **ralgb5a** у середньому приходиться близько трьох обчислень значення функції та її суперградієнта. Максимум (2460/739 = 3.33) реалізується при $n=1500$, $m=3000$, мінімум (1500/502 = 2.99) – при $n=500$, $m=1000$.

Другий обчислювальний експеримент назвемо МНМ-експериментом. Він пов'язаний з оцінкою за допомогою методу найменших модулів (МНМ) n невідомих параметрів $x=\left(x_1,..,x_n\right)^T\in\mathbf{R}^n$ для m спостережень $y=\left(y_1,..,y_m\right)^T\in\mathbf{R}^m$, які пов'язані співвідношенням:

$$y=Ax+u, \tag{7}$$

де $A=\left\{a_{ij}\right\}$ – матриця розміру $m\times n$, a_{ij} – відомі коефіцієнти, $u=\left(u_1,..,u_m\right)^T\in\mathbf{R}^m$ – невідомі випадкові величини, що мають (приблизно) однакові функції розподілу.

Методу найменших модулів (знаходяться компоненти невідомого вектора x^* згідно з критерієм найменших модулів) відповідає задача математичного програмування:

$$f^*_{LMM}=f_{LMM}\left(x^*\right)=\min_{x\in\mathbf{R}^n}\left\{f_{LMM}\left(x\right)=\sum_{i=1}^{m}\left|y_i-\sum_{j=1}^{n}a_{ij}x_j\right|\right\}, \tag{8}$$

де $|\cdot|$ – модуль (абсолютна величина) числа. Задача (8) – безумовна задача мінімізації опуклої кусково-лінійної функції. Зауважимо, що метод найменших модулів є робастним до аномальних спостережень або «викидів» [7].

Задача (8) – безумовна задача мінімізації опуклої кусково-лінійної функції $f_{LMM}\left(x\right)$, значення та субградієнт якої у точці $\bar{x}$ обчислюється за формулами octave-функції **fgLMM**:

```
function [f,g] = fgLMM(x)
global A y n m # parameters for problem (8)
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 (8) для octave-функції **fgLMM** є наступними: **A** – матриця відомих коефіцієнтів, **y** – вектор результатів спостережень, **n** – кількість параметрів, **m** – кількість спостережень.

Мета МНМ-експерименту – оцінка часу, необхідного програмі **ralgb5a** для розв'язання задачі (8) при $n \in \{200, 500, 1000, 1500, 2000\}$ та $m = 2n$. Вхідні дані для задач (7) та (8) генерувалися за формулами: **A = ones(m,n)+rand(m,n),** y = **A*ones(n,1)**. Параметри $r(\alpha)$-алгоритму для МНМ-експерименту вибиралися такими ж як і для першого ЛП-експерименту, за винятком $q_1 = 0.95$, $h_0 = 5.0$ та $\varepsilon_x = 10^{-6}$. МНМ-експеримент реалізує наведений далі octave-код.

```
#  computational efficiency of ralgb5a for problem (8)
global A y n m
printf("\n"); # set parameters for r-algorithm
alpha = 2.0, h0 = 5.0, q1 = 0.95,
epsx = 1.e-6, epsg = 1.e-8, maxitn = 2500, intp = 100,
nmtest = [ 200 400; 500 1000; 1000 2000; 1500 3000; 2000 4000];
rand("seed", 2022); 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 наведено в табл. 2, де f_r – значення функції, що мінімізується, колонки *time*, *itn*, *nfg*, *ist* аналогічні колонкам з табл. 1, $\|x_r - x^*\|$ – норма відхилення знайденого значення x_r від значення точки мінімуму $x^* = (1,1,\ldots,)^T$.

ТАБЛИЦЯ 2. МНМ-експеримент: затрати octave-функції **ralgb5a** для розв'язання тестових задач (8)

n	*m*	f_r	*time*	*itn*	*nfg*	*ist*	$\|x_r - x^*\|$
200	400	4.02e-05	0.20	295	303	3	6.05e-07
500	1000	1.60e-04	1.65	303	314	3	7.38e-07
1000	2000	4.86e-04	6.56	305	318	3	8.82e-07
1500	3000	5.61e-04	15.18	304	317	3	7.78e-07
2000	4000	7.64e-04	27.90	307	322	3	7.84e-07

З табл. 2 (колонка «*time*») видно, що за 10 – 30 секунд на сучасних ПЕОМ octave-реалізація **ralgb5a** дозволяє знаходити оцінки параметрів за методом найменших модулів для кількості параметрів $n=1000\div2000$ та кількості вимірювань $m=2000\div4000$. Так, наприклад, для тестового прикладу при $n=1500$, $m=3000$ затрачено 14 с, а при $n=2000$, $m=4000$ – 25 с. З колонки «$\|x_r - x^*\|$» бачимо, що розв'язок задачі (8) знайдено з великою точністю. При цьому на одну ітерацію *r(α)*-алгоритму приходиться одне обчислення значення функції та її субградієнта.

3. Python-функція ralgb5a. Вона реалізує описану в розділі 1 модифікацію *r(α)*-алгоритму з адаптивним регулюванням величини кроку та виконана у вигляді окремого модуля, який може бути підвантажений у довільну Python-програму та запущений із набором параметрів за замовчуванням. Функція **calcfg** для обчислення функції та субградієнту (суперградієнту) може приймати опційний набір параметрів у вигляді кортежу (порожній за замовчуванням). Коди зупинки та друк на консоль аналогічні тим, що використані для octave-функції **ralgb5a**.

Код python-функції **ralgb5a**, який включає англомовні коментарі для вхідних та вихідних параметрів, та код тестового прикладу для мінімізації функції $f(x)=\sum_{i=1}^{100}(x_i-i)^2$ наведено далі.

```
from numpy import eye, zeros, ones, round_, arange
from numpy.linalg import norm
from numpy.random import random
from time import time

def ralgb5a(calcfg, im, x, alpha=2.0, h0=1.0, q1=0.95, epsg=1e-7,
            epsx=1e-6, maxitn=2000, itnp=50, cfgargs=() ):
    """
    ralgb5a(calcfg, 1, x, *)

    Used to find the optimal (minimal or maximal) value of the given function.
    Returns the best extremal value found.

    Parameters
    ----------
    calcfg : function
        function which returns a tuple of the optimizing function and its
        gradient for the given vectorial point x (ndarray).
        For example: ``funval, gradient = calcfg(x, *cfgargs)``.
    im : integer
        should be 1 for minimization and -1 for maximization problems.
    x : ndarray
        starting point for the optimization problem. Should be a column
        vector.
    alpha : real, optional
        coefficient of space dilation. The default value is 2. Recommended
        to keep in between 2 and 4.
    h0 : real, optional
        parameter of the adaptive step. Default value is 1 (recommended).
    q1 : real, optional
        parameter of the adaptive step. Default value is 0.9. For smooth
        optimization problems it is recommended to keep q1 in between 0.8
        and 0.95. For nonsmooth optimization problem use q1 = 1.
    epsg : real, optional
        stopping criterion by gradient norm. Default value is 1e-7.
    epsx : real, optional
```

```
    stopping criterion by argument change norm. Default value is 1e-6.
maxitn : integer, optional
    stopping criterion by iterations number. Default value is 2000.
itnp : integer, optional
    interval to print information about iterations. Default value is
    50. Use negative value to disable output.
    Output:
    itn -- iteration number
    f -- function value
    fr -- best found function value
    nfg -- total number of calculations of the function value and
         gradient
    lsa -- number of steps along the gradient direction
    lsm -- maximal number of steps along the gradient direction
cfgargs : list, optional
    arguments for calcfg function to call it as
    ``calcfg(x, *cfgargs)``. Default value is [].

Returns
-------
xr : ndarray
    point corresponding to the best function value found
fr : real
    best function value found
itn : integer
    number of iterations
ist : integer
    exit code:
    2 -- by epsg
    3 -- by epsx
    4 -- by maxitn
    5 -- warning

"""
itn = 0
lsa = 0; lsm = 0
hs  = h0
B   = eye(len(x))
xr  = x
fr, g0 = calcfg(xr, *cfgargs); nfg = 1
if itnp > 0:
    print("itn {:5d} f {:15.5f} fr {:15.5f} nfg {:5d}".format(itn,
          fr, fr, nfg))

for itn in range(1, maxitn+1):
    g1 = B.transpose().dot(g0)
    dx = B.dot(g1)/norm(g1)
    d = 1; ls = 0; ddx = 0
    while d > 0:
        x -= im*hs*dx
        ddx += hs*norm(dx)
        [f, g1] = calcfg(x, *cfgargs); nfg += 1
        if im*f < im*fr:
            fr = f; xr = x.copy()
        if norm(g1) < epsg:
            ist = 2
            return xr, fr, itn, nfg, ist
```

```
            ls += 1
            if ls % 3 == 0:
                hs *= 1.1
            if ls > 500:
                ist = 5
                return xr, fr, itn, nfg, ist
            d = dx.transpose().dot(g1)
        if ls == 1:
            hs *= q1
        lsa += ls
        lsm = max(lsm, ls)
        if itn % itnp == 0:
            if itnp > 0:
                print("itn {:5d} f {:15.5e} fr {:15.5e} nfg {:5d} lsa {:5d}"
                      " lsm {:5d}".format(itn, f, fr, nfg, lsa, lsm))
            lsa = 0; lsm = 0
        if ddx < epsx:
            ist = 3
            return xr, fr, itn, nfg, ist
        dg = B.transpose().dot(g1 - g0)
        xi = dg/norm(dg)
        B += (1/alpha - 1)*B.dot(xi.dot(xi.transpose()))
        g0 = g1
    ist = 4
    return xr, fr, itn, nfg, ist

def testfun(x, n):
    """Calculates function and gradient for
    f(x) = \sum_{i=1}^n (x_i - i)^2
    """
    x0 = arange(1, n+1).reshape(n, 1)
    dx = x - x0
    return (dx**2).sum(), 2*dx

def main():
    n = 100
    x0 = -12*ones((n, 1))
    xstar = arange(1, n+1).reshape(n, 1)
    print('----- ralgb5a started -----')
    ts = time()
    xr, fr, itn, nfg, ist = ralgb5a(testfun, 1, x0, itnp=1, h0=250.0,
                                    cfgargs=[n])
    te = time()
    print('----- ralgb5a finished -----')
    print('n {:8d} | '
        'ist {:2d} | '
        'time {:3} sec'.format(n, ist, round_(te - ts, 3)))
    print('fr {:16.8e} | '
        'norm(xr-x*) {:12.4e} '.format(fr, norm(xr-xstar)))

if __name__ == '__main__':
    main()
```

Враховуючи, що стартова точка $x_0 = (-12,-12,\ldots,-12)^T$ є далекою від $x^* = (1,2,3,\ldots,100)^T$ – точки мінімуму функції $f(x) = \sum_{i=1}^{100}(x_i - i)^2$, то початковий крок h_0 вибирався рівним 250. В результаті роботи наведеного вище коду отримуємо такий протокол друку:

```
----- ralgb5a started -----
itn      0 f    473950.00000 fr    473950.00000 nfg      1
itn      1 f      3.78960e+03 fr     3.78960e+03 nfg      4 lsa      3 lsm      3
itn      2 f      5.76692e+03 fr     3.78960e+03 nfg      5 lsa      1 lsm      1
itn      3 f      2.28577e+03 fr     1.97832e+02 nfg      7 lsa      2 lsm      2
itn      4 f      1.97832e+02 fr     1.97832e+02 nfg      9 lsa      2 lsm      2
itn      5 f      1.96977e+00 fr     1.96977e+00 nfg     10 lsa      1 lsm      1

itn    150 f      1.79811e-03 fr     5.39184e-04 nfg    263 lsa      2 lsm      2
itn    151 f      3.42515e-02 fr     5.39184e-04 nfg    264 lsa      1 lsm      1
itn    152 f      5.07642e-04 fr     5.07642e-04 nfg    266 lsa      2 lsm      2

itn    250 f      1.50265e-03 fr     1.09507e-06 nfg    428 lsa      1 lsm      1
itn    251 f      1.78924e-06 fr     1.09507e-06 nfg    430 lsa      2 lsm      2
itn    252 f      5.81046e-05 fr     1.09507e-06 nfg    431 lsa      1 lsm      1

itn    428 f      1.07259e-12 fr     2.91648e-14 nfg    701 lsa      2 lsm      2
itn    429 f      6.16558e-14 fr     2.91648e-14 nfg    703 lsa      2 lsm      2
itn    430 f      4.79000e-11 fr     2.91648e-14 nfg    704 lsa      1 lsm      1
itn    431 f      5.15732e-12 fr     2.91648e-14 nfg    706 lsa      2 lsm      2
----- ralgb5a finished -----
n       100 | ist  2 | time 0.135 sec
fr   8.57568295e-16 | norm(xr-x*)   2.9284e-08
```

Порівняння часових витрат на виконання тестових задач з ЛП- та МНМ-експериментів у середовищах Octave та Python наведено в табл. 3, де t_{oct} – час (в секундах), який витратила octave-функція **ralgb5a** на розв'язання задач (6) та (8), t_{py} – час, який витратила python-функція **ralgb5a**. Розрахунки проводилися на комп'ютері з процесором Intel(R) Core(TM) i7-9700 з тактовою частотою 3,00 ГГц та 32 ГБ оперативної пам'яті за допомогою GNU Octave версії 6.4.0 та Python 3.10.6 з бібліотекою матричних обчислень NumPy версії 1.22.0. Незначні відмінності в кількостях ітерацій та в кількостях обчислень значень функцій та їх супер- та суб-градієнтів (порівняно з табл.1 та 2) зумовлені тим, що вхідні дані задач генерувались окремо і передавались програмам за допомогою зовнішніх файлів та тим, що випадкові числа генерувалися під різними версіями GNU Octave.

ТАБЛИЦЯ 3. ЛП- та МНМ-експерименти: затрати python-функції **ralgb5a** для розв'язання тестових задач (6), (8)

n	*m*	ЛП-експеримент				МНМ-експеримент			
		t_{oct}	t_{py}	*itn*	*nfg*	t_{oct}	t_{py}	*itn*	*nfg*
200	400	0.57	0.52	823	2602	0.12	0.19	293	301
500	1000	1.63	3.44	830	2749	0.74	1.51	303	314
1000	2000	6.97	11.0	705	2300	5.08	9.04	305	318
1500	3000	19.9	36.2	806	2713	13.9	26.7	305	318
2000	4000	29.8	67.8	676	2209	25.0	49.9	307	322

Результати обчислювальних експериментів з негладкими функціями (6) та (8) показують, що стандартна конфігурація Python-програми сповільнюється на 30 – 50% у порівнянні з Octave-

програмою при збільшенні розміру задачі до кількох тисяч змінних, показуючи однакові результати по кількості ітерацій (*itn* та *nfg*) за умови однакових вхідних даних. Використання обгортки FlexiBLAS дозволило встановити, що реалізації OpenBLAS на бібліотеках Threads і OpenMP показують практично однакові результати на восьми віртуальних процесорах. Зауважимо, що реалізація Netlib, яка не є паралельною, виконує матричні обчислення в 30 разів повільніше.

На рисунку показано графік залежності кількості ітерацій, необхідних для розв'язання задачі (8) з $n=500$ та $m=1000$, від вибору коефіцієнта розтягу простору α з діапазону від $\alpha=2$ до $\alpha=4$ та коефіцієнта зменшення кроку q_1 з діапазону від $q_1=0.8$ до $q_1=1.0$. Найкраще значення (54 ітерації та 72 обчислення функції та субградієнту) відмічено червоним еліпсом, якому відповідає використання $\alpha=2.6$ та $q_1=0.81$.

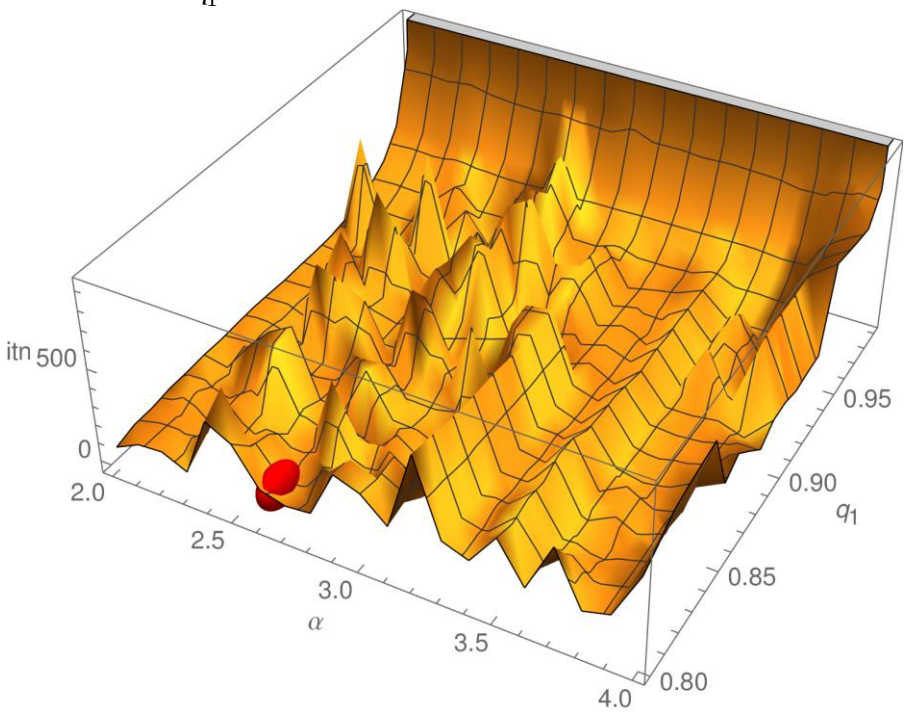

РИСУНОК. Кількість ітерацій для розв'язання задачі (8) з $n=500$ та $m=1000$ для МНМ-експерименту

З рисунку видно, що для малих значень α кількість ітерацій монотонно зростає з наближенням q_1 до одиниці. Для $2.3\le\alpha\le3.3$ кількість ітерацій як функція α і q_1 стає сильно немонотонною, проте дозволяє знайти розв'язок за 72 обчислення функції та градієнту у порівнянні з 90 обчисленнями для $\alpha=2$. Для $\alpha\ge3.3$ відновлюється монотонне зростання кількості ітерацій, якщо q_1 наближується до одиниці.

Якщо найкраще значення $\alpha=2.6$ та $q_1=0.81$ (див. рис. 1) використати для розв'язання задачі (8) при $n=5000$ та $m=10000$, то для знаходження точки x_r – такої, що $\left\|x_r-x^*\right\|\le2.5\times10^{-6}$, octave-функція **ralgb5a** затрачує 102 секунди (140 ітерацій та 219 обчислень функції та її субградієнта). При цьому значення функції зменшується на 10 порядків – від $f(x_0)=7.499750\times10^{+7}$ до $f(x_r)=5.248631\times10^{-3}$. Використовувалися $x_0=0$, $h_0=5.0$ та параметри зупинки: $\varepsilon_x=10^{-6}$,

$\varepsilon_g = 10^{-8}$. Обчислення проводились на комп'ютері з процесором Intel Core i5-9400f з тактовою частотою 2,9 ГГц та 16 ГБ оперативної пам'яті з використанням GNU Octave версії 5.1.0.

Висновки. У статті описано дві програмні реалізації модифікації *r*-алгоритму Шора з постійним коефіцієнтом розтягу простору та адаптивним регулюванням кроку: перша програма реалізована на мові GNU Octave, а друга програма реалізована мовою Python.

Для задачі мінімізації опуклої функції описано модифікацію *r*-алгоритму з постійним коефіцієнтом розтягу простору в напрямку різниці двох послідовних субградієнтів та адаптивним способом регулювання кроку у напрямку антисубградієнта у перетвореному просторі змінних. Представлено програмні реалізації цієї модифікації у вигляді octave-функції **ralgb5a** та python-функції **ralgb5a**, які дозволяють знаходити або наближення до точки мінімуму опуклої функції, або наближення до точки максимуму увігнутої функції. Наведено коди обох функцій з коротким описом їх вхідних та вихідних параметрів.

Описано тестові експерименти для дослідження ефективності функцій **ralgb5a** для максимізації кусково-лінійної увігнутої функції, яка отримана за допомогою методу негладких штрафних функцій для задачі лінійного програмування, та мінімізації кусково-лінійної опуклої функції, яка відповідає методу найменших модулів. Представлено результати обчислювальних експериментів для розв'язання тестових задач з 200, 500, 1000, 1500 та 2000 змінних, які демонструють ефективну роботу функцій **ralgb5a**. Представлено результати обчислювального експерименту для розв'язання тестової задачі за методом найменших модулів для 5000 змінних та 10000 спостережень.

Розглянуті програмні реалізації можна використати у різноманітних економічних застосуваннях, у тому числі в задачах кредитного скорингу [12], де вимагається розв'язання задач лінійного програмування та регресійного аналізу. Якщо кількість невідомих параметрів буде не більша 2000, то для розв'язання відповідних задач знадобиться менше однієї хвилини для сучасних ПЕОМ.

Робота підтримана грантами: CRDF Global (грант G-202102-68020) та Volkswagen Foundation (грант № 97775).

Список літератури

1. Шор Н.З. Методы минимизации недифференцируемых функций и их приложения: автореф. дис. ... докт. физ-мат. наук. Киев, 1970. 44 с.
2. Шор Н.З., Журбенко Н.Г. Метод минимизации, использующий операцию растяжения пространства в направлении разности двух последовательных градиентов. *Кибернетика.* 1971. 3. С. 51–59.
3. Шор Н.З. Методы минимизации недифференцируемых функций и их приложения. Киев: Наукова думка, 1979. 200 с.
4. Шор Н.З., Стеценко С.И. Квадратичные экстремальные задачи и недифференцируемая оптимизация. Киев: Наук. думка, 1989. 208 с.
5. Shor N.Z. Non-Differentiable Optimization and Polynomial Problems. Kluwer Academic Publishers, Boston, Dordrecht, London. 1998. 412 p.
6. Eaton J.W., Bateman D., Hauberg S. GNU Octave Manual Version 3. Network Theory Ltd. 2008. 568 p.
7. Стецюк П.И. Теория и программные реализации *r*-алгоритмов Шора. *Кибернетика и системный анализ*. 2017. № 5. С. 43–57.
8. Стецюк П.И. Субградиентные методы ralgb5 и ralgb4 для минимизации овражных выпуклых функций. *Вычислительные технологии*. 2017. Т. 22, № 2. С. 127–149.
9. Стецюк П.І. Комп'ютерна програма "Octave-програма ralgb5a: *r*(α)-алгоритм з адаптивним кроком". Свідоцтво про реєстрацію авторського права на твір № 85010. Україна. Міністерство економічного розвитку і торгівлі. Державний департамент інтелектуальної власності. Дата реєстрації 29.01.2019.
10. 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.
11. Хьюбер Дж. П. Робастность в статистике. М.: Мир, 1984. 304 с.
12. Lyn C. Thomas, David B. Edelman, Jonathan N. Crook. Credit Scoring and its Applications. SIAM Monographs on Mathematical Modeling and Computation. Philadelphia, 2002. 243 p.

Одержано 16.09.2022

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

Пилиповський Олександр Васильович,
кандидат фізико-математичних наук, молодший науковий співробітник
Київського академічного університету, Київ,
o.pylypovskyi@kau.edu.ua
https://orcid.org/0000-0002-5947-9760

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

UDC 519.85

Petro Stetsyuk [1 *], Aleksandr Pylypovskyi [2], Olha Khomiak [1]

GNU Octave and Python Implementation of Shor's *r*-Algorithm with Adaptive Step Control

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

r-algorithms, or subgradient methods with dilation of space in the direction of the difference of two sequential subgradients, were proposed by N.Z.Shor in 1970 in his doctoral thesis. Respective software implementations proved to be competitive with the most effective methods for smooth ill-conditioned problems, both in terms of reliability and calculation time and accuracy of results.

The article is devoted to the description of two software implementations of Shor's *r*-algorithm modification with a constant coefficient of space dilation and adaptive step control. The first program is implemented using the GNU Octave, and the second program is implemented using Python.

Material of the paper is presented in three sections. In the first section, we describe a modification of the *r*-algorithm with a constant coefficient of space dilation in the direction of the difference of two sequential 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 either 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.

The second section describes test experiments to investigate efficiency of the octave-function ralgb5a to maximizing the piecewise linear concave function, which is obtained using the method of non-smooth penalty functions for the linear programming problem. Another example represents minimization of the piecewise linear convex function, which corresponds to the method of least modules. Results of these computational experiments for test problems with 200, 500, 1000, 1500 and 2000 variables are presented to demonstrate the effective operation of the octave-function ralgb5a.

The third section describes the python function ralgb5a and provides its code with a description of the input and output parameters. It is show, how the ralgb5a function can be accelerated by setting two parameters. The results of computational experiments to solve the test problem using the method of least modules for 5,000 variables and 10,000 observations are presented.

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