ПРОГРАММА ralgb5 ДЛЯ МИНИМИЗАЦИИ ОВРАЖНЫХ ВЫПУКЛЫХ ФУНКЦИЙ

Стецюк П.И., stetsyukp@gmail.com
Институт кибернетики им. В.М. Глушкова НАН Украины

Введение. Субградиентные методы, в которых используется растяжение пространства в направлении разности двух последовательных субградиентов, предложены в [1, 2]. Они получили название r-алгоритмов (от русского слова "разность") и стали одним из центральных результатов докторской диссертации Н.З. Шора (1970). Программные реализации r-алгоритмов оказались вполне конкурентноспособны как по надежности так и по времени счета и точности результатов с наиболее эффективными методами решения гладких плохо обусловленных задач. Ускоренную сходимость r-алгоритмов при минимизации негладких выпуклых функций обеспечивает сочетание в них двух тесно связанных между собой идей.

Первая идея состоит в использовании процедуры наискорейшего спуска в направлении антисубградиента выпуклой функции в преобразованном пространстве переменных. Она гарантирует монотонность по значениям выпуклой функции для точек минимизирующей последовательности, которая конструируется r-алгоритмами. Если поиск минимума функции в направлении антисубградиента осуществляется приближенно, то тогда "монотонность" по минимизируемой функции заменяется на "почти монотонность". Вторая идея состоит в использовании операции растяжения пространства в направлении разности двух последовательных субградиентов, где второй субградиент вычислен в точке минимума функции по направлению первого антисубградиента. В результате этого растяжения уменьшаются поперечные составляющие субградиентов вдоль направления на точку минимума, что позволяет уменьшить степень "вытянутости" овражной функции в преобразованном пространстве переменных.

В статье опишем субградиентный метод **ralgb5**, который является одной из реализаций r-алгоритмов с адаптивной регулировкой шага в направлении нормированного антисубградиента в преобразованном пространстве переменных и постоянным коэффициентом растяжения пространства. Метод **ralgb5** реализован одноименной функцией на языке **octave** [3] и базируется на устойчивой B-форме r-алгоритмов, где B – невырожденная $n\times n$-матрица. Его название связано с тем, что коррекция матрицы B и другие вычисления на каждой итерации метода требуют $5n^2$ арифметических операций умножения.

Статья содержит три раздела. В первом разделе описана вычислительная форма $r(\alpha)$-алгоритмов и проанализированы ее свойства. Во втором разделе описаны $r(\alpha)$-алгоритм с адаптивной регулировкой шага и имеющиеся его программные реализации. В третьем разделе описана octave-функция **ralgb5** и приведен ее код.

1. $r(\alpha)$-алгоритм. Пусть $f(x)$ – выпуклая функция, где x – вектор из n переменных. Минимальное значение функции $f(x)$ будем обозначать $f^* = f(x^*)$, $x^* \in X^*$. Будем предполагать, что $f(x)$ имеет ограниченное множество минимумов X^*, т.е. выполняется условие $\lim\limits_{\|x\|\to\infty} f(x) = +\infty$. Это условие обеспечивает корректность регулировки шага в r-алгоритмах. Обозначим α –коэффициент растяжения пространства, такой что $\alpha > 1$.

$r(\alpha)$-Алгоритмом для минимизации $f(x)$ называется итеративная процедура нахождения последовательности n-мерных векторов $\{x_k\}_{k=0}^{\infty}$ и последовательности $n\times n$-матриц $\{B_k\}_{k=0}^{\infty}$ по следующему правилу:

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

где

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

$$\eta_k = \frac{B_k^T r_k}{\left\| B_k^T r_k \right\|}, \quad r_k = g_f(x_{k+1}) - g_f(x_k), \quad \beta = \frac{1}{\alpha} < 1. \qquad (3)$$

Здесь x_0 – стартовая точка; $B_0 = I_n$ – единичная $n \times n$-матрица[1]; h_k^* – величина шага из условия минимума функции $f(x)$ в направлении нормированного антисубградиента в преобразованном пространстве переменных; $R_\beta(\eta) = I_n + (\beta - 1)\eta\eta^T$ – оператор сжатия пространства субградиентов в нормированном направлении η с коэффициентом $\beta = \frac{1}{\alpha} < 1$; $g_f(x_k)$ и $g_f(x_{k+1})$ – субградиенты функции $f(x)$ в точках x_k и x_{k+1}. Если $g_f(x_k) = 0$, то $x_k = x^*$ и процесс (1)–(3) останавливается.

Комментарий. На каждой итерации r-алгоритмов реализуется субградиентный спуск для выпуклой функции $\varphi(y) = f(B_k y)$ в преобразованном пространстве переменных $y = A_k x$, где $A_k = B_k^{-1}$. На самом деле, если обе части формулы $x_{k+1} = x_k - h_k B_k \xi_k$ умножить слева на матрицу A_k, то получим

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

где вектор $g_\varphi(y_k) = B_k^T g_f(x_k)$ является субградиентом функции $\varphi(y) = f(B_k y)$ в точке $y_k = A_k x_k$ пространства переменных $y = A_k x$. Это легко видеть из того, что субградиент функции $f(x)$ в точке x_k удовлетворяет неравенству

$$f(x) \geq f(x_k) + (g_f(x_k))^T (x - x_k) \quad \forall x \in E^n,$$

откуда, осуществляя замену переменных $x = B_k y$, получаем

$$\varphi(y) \geq \varphi(y_k) + (B_k^T g_f(x_k))^T (y - y_k) = \varphi(y_k) + (g_\varphi(y_k))^T (y - y_k) \quad \forall y \in E^n.$$

[1] В качестве матрицы B_0 часто выбирают диагональную матрицу D_n с положительными коэффициентами по диагонали, с помощью которой осуществляется масштабирование переменных.

Если $h_k = h_k^*$, то формула (4) означает точный поиск минимума функции $\varphi(y) = f(B_k y)$ в направлении нормированного антисубградиента в преобразованном пространстве переменных $y = A_k x$, а если $h_k \approx h_k^*$, то – приближенный поиск. Если функция $f(x)$ является недифференцируемой в точке x_k, то возможен случай $h_k = h_k^* = 0$, с которым связаны основные проблемы с условием останова r-алгоритмов для негладких функций.

Метод (1)–(3) называют B-формой r-алгоритмов; на каждой его итерации корректируется матрица, связанная с заменой переменных $x = By$. Итерация метода требует порядка $5n^2$ арифметических операций умножения, которые определяют вычислительную трудоемкость итерации (операции сложения учитывать не будем из-за их малого вклада в трудоемкость итерации). Из них $3n^2$ умножений требуется для вычисления векторов $B_k\xi_k$, $B_k^T g_f(x_k)$ и $B_k^T r_k$ (умножение матрицы на вектор), а $2n^2$ умножений требует одноранговый пересчет матрицы $B_{k+1} = B_k R_{\beta_k}(\eta_k)$. Действительно,

$$B_{k+1} = B_k R_{\beta_k}(\eta_k) = B_k(I_n + (\beta_k - 1)\eta_k\eta_k^T) = B_k + (\beta_k - 1)(B_k\eta_k)\eta_k^T,$$

откуда легко видеть, что вычисление вектора $\eta = B_k\eta_k$ требует n^2 умножений и столько же умножений требует построение одноранговой матрицы $\eta\eta_k^T$.

2. $r(\alpha)$-Алгоритм с адаптивным шагом. Одним из эффективных зарекомендовал себя $r(\alpha)$-алгоритм с адаптивной регулировкой шага. Это вариант r-алгоритмов, в котором величина шага h_k настраивается в процессе выполнения одномерного спуска в направлении нормированного антисубградиента в преобразованном пространстве переменных с помощью параметров h_0, q_1, n_h, q_2. Здесь h_0 – величина начального шага (используется на 1-й итерации, на каждой последующей итерации эта величина уточняется); q_1 – коэффициент уменьшения шага ($q_1 \le 1$), если условие завершения спуска по направлению выполняется за один шаг; q_2 –

коэффициент увеличения шага ($q_2 \geq 1$); натуральное число n_h задает число шагов одномерного спуска ($n_h > 1$), через каждые из которых шаг будет увеличиваться в q_2 раз.

Условие завершения спуска по направлению выполняется как только обнаружена точка x_{k+1}, для которой $(x_{k+1} - x_k)^T g_f(x_{k+1}) \geq 0$, и легко проверяется, так как в силу положительности шага равносильно выполнению неравенства $(B_k B_k^T g_f(x_k))^T g_f(x_{k+1}) \leq 0$. Последнее неравенство означает, что угол между двумя последовательными субградиентами в преобразованном пространстве переменных будет неострым. Действительно, его можно записать как неравенство $(B_k^T g_f(x_k))^T B_k^T g_f(x_{k+1}) \leq 0$, которое равносильно неравенству $(g_\varphi(y_k))^T g_\varphi(y_{k+1}) \leq 0$, где $g_\varphi(y_k) = B_k^T g_f(x_k)$ и $g_\varphi(y_{k+1}) = B_k^T g_f(x_{k+1})$ являются субградиентами функции $\varphi(y) = f(B_k y)$ в точках $y_k = A_k x_k$ и $y_{k+1} = A_k x_{k+1}$ преобразованного пространства переменных $y = A_k x$, где $A_k = B_k^{-1}$. Поскольку предполагается, что $\lim\limits_{\|x\| \to \infty} f(x) = +\infty$, то после конечного числа шагов адаптивного спуска в направлении нормированного антисубградиента обязательно выполнится условие завершения спуска.

Итеративный процесс в $r(\alpha)$-алгоритме с адаптивной регулировкой шага продолжается до выполнения некоторого критерия останова, где ключевую роль играют параметры ε_x и ε_g. Они определяют два условия останова: метод останавливается в точке x_{k+1}, если выполнено условие $\|x_{k+1} - x_k\| \leq \varepsilon_x$ (останов по аргументу); метод останавливается в точке x_{k+1}, если выполнено условие $\|g_f(x_{k+1})\| \leq \varepsilon_g$ (останов по норме субградиента, используется для гладких функций). Кроме них используются еще два условия останова метода: стандартный останов, если превышено максимальное количество итераций, и аварийный останов, который

сигнализирует о том, что либо функция $f(x)$ неограничена снизу, либо начальный шаг h_0 слишком мал и его требуется увеличить. И хотя $r(\alpha)$-алгоритм с адаптивной регулировкой шага не гарантирует монотонного убывания функции, однако, как показали эксперименты, возрастание функции происходит достаточно редко. Подробные рекомендации по выбору коэффициента растяжения пространства и параметров адаптивной регулировки шага можно найти в [4], стр. 45–47. Суть их выбора состоит в том, чтобы адаптивный способ регулировки шага позволял увеличивать точность поиска минимума функции по направлению в процессе счета и при этом число шагов по направлению не должно превышать в среднем двух-трех на одну итерацию.

Если $r(\alpha)$-алгоритм с адаптивной регулировкой шага применять для минимизации негладких функций, то рекомендуется следующий выбор параметров: $\alpha = 2 \div 4$, $h_0 = 1.0$, $q_1 = 1.0$, $q_2 = 1.1 \div 1.2$, $n_h = 2 \div 3$. Если известна априорная оценка расстояния от начальной точки x_0 до точки минимума x^*, то начальный шаг h_0 целесообразно выбирать порядка $\|x_0 - x^*\|$. При минимизации гладких функций рекомендуемые параметры такие же, за исключением q_1 ($q_1 = 0.8 \div 0.95$). Это обусловлено тем, что дополнительное измельчение шага способствует увеличению точности поиска минимума функции по направлению, что при минимизации гладких функций обеспечивает более быструю скорость сходимости. При таком выборе параметров, как правило, число спусков по направлению редко превосходит два, а за n шагов точность по функции улучшается в три-пять раз. Параметры останова $\varepsilon_x, \varepsilon_g \sim 10^{-6} \div 10^{-5}$ при минимизации выпуклой функции, даже существенно овражной структуры, обеспечивают нахождение точки x_r^* – приближения к точке x^*, для которого значение функции, достаточно близко к оптимальному: $\dfrac{f(x_r^*) - f^*}{|f^*| + 1} \sim 10^{-6} \div 10^{-5}$ – для негладких

и $\frac{f(x_r^*) - f^*}{|f^*|+1} \sim 10^{-12} \div 10^{-10}$ – для гладких функций. Это подтверждается результатами многочисленных тестовых и реальных расчетов.

В настоящее время $r(\alpha)$-алгоритм с адаптивной регулировкой шага и ее модификациями реализован рядом компьютерных программ на языках программирования Фортран, Си, C++, C# и octave. На основе различных модификаций метода (1)–(3) разработаны компьютерные программы **ralg** (Фортран,Си, C++), **ralgb4** (Фортран), **SolveOpt** (Фортран, Си). На основе B-формы, которая использует метод (1)–(3), построена компьютерная программа **ralgb5** (Фортран, octave, C++ и C#).

Исторически одной из первых была фортрановская программа **ralg** (автор Н.Г. Журбенко) для решения вычислительных задач на компьютерах БЭСМ-6 и ЕС-1040 [5]. В 70–80 гг. прошлого века программа **ralg** активно использовалась как сотрудниками и аспирантами Института кибернетики АН Украины, так и сотрудниками других организаций, которые сталкивались с задачами оптимизации негладких функций. В 90-ые годы программа **ralg** послужила основой для программной реализации фортрановской программы **ralgb4** (автор П.И. Стецюк), которая использует модификацию r-алгоритма из [6]. С помощью программы **ralg** А.В. Кунцевич разработал комплекс программ **SolveOpt** (языки Фортран и Си), где использована усложненная адаптивная регулировка шага и критерии останова $|x_{k+1}^i - x_k^i| \le \delta_x |x_{k+1}^i|$ и $|f(x_{k+1}) - f(x_k)| \le \delta_f |f(x_{k+1})|$ с заданными достаточно малыми δ_x и δ_f [7]. В конце прошлого – начале этого века программа **ralg** послужила основой для разработки компьютерных программ на языках программирования Си, C++ (авторы Н.Г. Журбенко и А.П. Лиховид).

В 2007-2008 гг. была разработана и протестирована фортрановская программа **ralgb5** (автор П.И. Стецюк), которая использует метод (1)–(3). Ее название связано с тем, что в основу программы положена B-форма r-алгоритма, которая требует $5n^2$ арифметических операций умножения на

каждой итерации. Фортрановский код программы **ralgb5** приведен на стр. 34–38 отчета [8], где программа использовалась при решении задач нелинейного программирования для нахождения оптимальных нагрузок энергоблоков тепловых электростанций при планировании почасового суточного графика потребления электрической энергии. В 2010 году программа **ralgb5** была переписана на язык octave, ее код приведен в [9], с. 384–385. Octave-программа **ralgb5** оказалась быстрее фортрановской программы, если речь шла о решении задач для тысячи и более переменных. Это было обусловлено тем, что библиотека BLAS для языка octave позволяет быстрее выполнять матрично-векторные операции в r-алгоритмах, чем это позволяет исполняемый код при оптимизирующих опциях компилятора Фортрана. В настоящее время программа **ralgb5** переведена А.П. Лиховидом на языки C++, C# и используется в программных имплементациях алгоритмов решения различного рода задач нелинейного программирования.

3. Octave-функция ralgb5. Программа **ralgb5** находит точку x_r^* – приближение к точке минимума выпуклой функции $f(x)$ от n переменных, которая для $r(\alpha)$-алгоритма с адаптивным шагом определяется выбранными: стартовой точкой x_0; коэффициентом растяжения пространства α, параметрами адаптивной регулировки шага h_0, n_h, q_1 и q_2; параметрами останова ε_g, ε_x и **maxitn**. Программа использует octave-функцию вида **function [f,g] = calcfg(x)**, которая вычисляет значение функции $f = f(x)$ и ее субградиента $g = g_f(x)$ в точке x. Имя функции вида **calcfg(x)** может быть произвольным, которое допускает синтаксис языка octave. Программа **ralgb5** использует следующие входные и выходные параметры.

```
%  Входные параметры:
%   calcfg -- имя функции вида calcfg(x) для вычисления f и g
%    x  -- начальная точка x0(1:n)  (на выходе портится)
%    alpha -- коэффициент растяжения пространства
%    h0,nh,q1,q2 -- параметры адаптивной регулировки шага
%    epsx,epsg,maxitn -- параметры останова
%   Выходные параметры:
%    xr -- найденная точка минимума функции xr(1:n)
%    fr -- значение функции f в точке xr
%    itn -- количество затраченных итераций
```

```
%    ncalls -- количество вызовов функции calcfg
%   istop -- код останова (2 = epsg, 3 = epsx, 4 = maxitn, 5 = error)
```

За точку x_r принимается такая точка итерационного процесса, реализующего $r(\alpha)$-алгоритм с адаптивным шагом, в которой получено наименьшее (рекордное) значение функции $f_r = f(x_r)$. Она не обязательно будет совпадать с последней точкой итерационного процесса, а может быть получена при одномерном поиске минимума функции по направлению на какой-либо из предыдущих итераций процесса. Статус точки x_r определяется кодом останова **istop** на итерации **itn = k**:

1. останов произошел по условию $\|g_f(\tilde{x}_k)\| \le \varepsilon_g$, где $\tilde{x}_k \in [x_k, x_{k+1}]$ (здесь $x_r = \tilde{x}_k$ и при малых ε_g функция $f(x)$ дифференцируема в рекордной точке x_r);
2. останов произошел по условию $\|x_{k+1} - x_k\| \le \varepsilon_x$ ($x_r = x_{k+1}$, если $f(x_{k+1})$ меньше, чем значения функции в остальных точках итерационного процесса);
3. останов произошел по условию **itn>maxitn** (превышено максимальное количество итераций и близость точки x_r к оптимальной x^* можно оценить по отклонениям f_r от значений функции на последних итерациях);
4. останов произошел из-за того, что за **500** шагов по направлению не выполнено условие завершения спуска (этот останов считается аварийным).

Аварийный останов связан либо с тем, что функция $f(x)$ неограничена снизу, либо начальный шаг h_0 слишком мал и его требуется увеличить. Это может быть как из-за того, что для выпуклой функции $f(x)$ не выполняется условие $\lim\limits_{\|x\|\to\infty} f(x) = +\infty$, так и из-за ошибок в программе вычисления значения функции и субградиента. Ошибка при вычислении функции менее

опасна (она влияет только на определение рекордной точки x_r), чем ошибка при вычислении компонент субградиента (градиента), который определяет выбор направления одномерного спуска. Аварийный останов из-за малости величины шага h_0 маловероятен, так как **500** шагов для спуска по направлению выбрано из тех соображений, чтобы для рекомендуемого минимального значения $q_2 = 1.1$ начальный шаг для очередной итерации мог максимально увеличиться в 10^6 раз, если $n_h = 3$, и в 10^{10} раз, если $n_h = 2$.

Программа **ralgb5** оформлена как octave-функция и ее код есть следующим:

```
# Octave-function ralgb5 for Shor's r-algorithm
function [xr,fr,itn,ncalls,istop] = ralgb5(calcfg,x,alpha,h0,q1, # row001
                                           q2,nh,epsg,epsx,maxitn);
itn = 0; hs = h0; B = eye(length(x)); xr = x;                    # row002
ncalls = 1; [fr,g0] = calcfg(xr);                                # row003
printf("itn %4d f %14.6e fr %14.6e ls %2d ncalls %4d\n",         # row004
       itn, fr, fr, 0, ncalls);
if(norm(g0) < epsg) istop = 2;  return; endif                    # row005
for (itn = 1:maxitn)                                             # row006
   dx = B * (g1 = B' * g0)/norm(g1);                             # row007
   d = 1; ls = 0; ddx = 0;                                       # row008
   while (d > 0)                                                 # row009
      x -= hs * dx; ddx += hs * norm(dx);                        # row010
      ncalls ++; [f, g1] = calcfg(x);                            # row011
      if (f < fr)  fr = f; xr = x; endif                         # row012
      if(norm(g1) < epsg) istop = 2;  return; endif              # row013
      ls ++; (mod(ls,nh)==0) && (hs *= q2);                      # row014
      if(ls > 500) istop = 5;  return; endif                     # row015
      d = dx' * g1;                                              # row016
   endwhile                                                      # row017
   (ls == 1) && (hs *= q1);                                      # row018
   printf("itn %4d f %14.6e fr %14.6e ls %2d ncalls %4d\n",      # row019
          itn, f, fr, ls, ncalls);
   if(ddx < epsx) istop = 3;  return; endif                      # row020
   xi = (dg = B' * (g1 - g0) )/norm(dg);                         # row021
   B += (1 / alpha - 1) * B * xi * xi';                          # row022
   g0 = g1;                                                      # row023
endfor                                                           # row024
istop = 4;                                                       # row025
endfunction                                                      # row026
```

Приведенный код состоит из **26** строк и большая часть из них содержит больше одного оператора языка **octave**. Функционально он полностью совпадает с кодом программы **ralgb5**, который приведен в [9], с. 384–385, и отличается от него только нумерацией строк и незначительным переформатированием текста.

Код octave-функции **ralgb5** реализован по формулам метода (1)–(3) так что итерационный процесс выполняется в цикле **for** (строки **6–24**), где для k-ой итерации субградиент $g_f(x_k)$ запоминается вектор-столбцом **g0**, а субградиент $g_f(x_{k+1})$ запоминается вектор-столбцом **g1**. Адаптивный способ регулировки шага на итерации реализуется строками **8–18** на основе внутреннего цикла **while**, который заканчивается как только выполняется условие завершения спуска по направлению, вычисленному в строке **7**. В цикле **while** обновляются "`рекордные"' точка и значение функции (строка **12**) и проверяются два условия останова: останов по норме градиента ε_g (строка **13**) и аварийный останов (строка **14**). Строка **19** обеспечивает печать на каждой итерации ее номера **itn**, значения функции **f**, рекордного значения функции **fr**, количества шагов одномерного спуска на итерации **ls** и полного количества вызовов программы для вычисления значений функции и ее субградиента **ncalls**. В строке **20** реализуется условие останова по аргументу, а в строках **21–23** реализуются пересчет матрицы B и установка суградиента **g0** для перехода к следующей итерации. В строках **2–5** инициализируются величины, необходимые для старта основного цикла **for** по итерациям (строка **6**).

Простота и прозрачность кода octave-функции **ralgb5** позволяет на ее основе легко разрабатывать оптимизационные ядра на языках **octave** и **MATLAB** для решения вычислительных задач, которые сводятся к проблемам минимизации негладких выпуклых функций или гладких выпуклых функций с овражной структурой поверхностей уровня. При использовании языка **MATLAB** в программе **ralgb5** потребуются модификации тех операторов языка **octave**, которых нет в **MATLAB**е. В коде octave-функции **ralgb5** это касается двух операторов из строк 7 и 21. Если строки 7 и 21 заменить на строки

```
g1 = B' * g0; g1 = g1 / norm(g1); dx = B * g1;          # row007a
dg = B' * (g1 - g0); xi = dg / norm(dg);                # row021a
```

то модифицированный код функции **ralgb5** будет идентичен коду на языке **MATLAB**.

Заключение. Субградиентный метод **ralgb5** можно использовать при решении негладких задач из различных областей приложений. Так как гладкая функция с очень быстро изменяющимся градиентом близка по своим свойствам к негладкой функции, то он обладает ускоренной сходимостью при оптимизации овражных гладких функций. Octave-функцию **ralgb5** можно использовать как оптимизационное ядро при реализации на языке octave алгоритмов решения задач нелинейного программирования. На ее основе легко разрабатывать оптимизационные ядра на языке **MATLAB** для решения вычислительных задач, которые сводятся к проблемам минимизации негладких выпуклых функций или гладких выпуклых функций с овражной структурой поверхностей уровня.

Octave-функцию **ralgb5** легко переписать на языки Фортран и Си, используя библиотеку базовых подпрограмм линейной алгебры BLAS (Basic Linear Algebra Subprograms) или библиотеку математических прикладных программ IntelR Math Kernel Library (IntelR MKL), которые оптимизированы под современные вычислительные машины. Это может позволить значительно ускорить методы для решения больших задач (тысяча и более переменных). Так, например, одну и ту же задачу с 1000 переменными программа **ralgb5** для GNU Octave версии 3.6.4 с использованием библиотеки BLAS решает в 2 раза быстрее, чем исполняемый оптимизируемый код компилятора Visual Studio C++ 2008 Express Edition без использования библиотеки BLAS. Эксперимент проводился на компьютере Pentium 3GHz в системе Windows7/32.

С помощью программы **ralgb5** можно находить достаточно точные приближения к точке минимума выпуклой функции. Если коэффициент растяжения пространства выбрать таким, чтобы он хорошо согласовывался с параметрами адаптивной регулировки шага в направлении нормированного антисубградиента в преобразованном пространстве переменных, то для

выполнения одних и тех же критериев останова можно значительно сократить количество итераций и количество вычислений значения функции и субградиента(градиента). Это зависит от конкретного вида минимизируемой функции, степени ее овражности и масштаба переменных.

Работа выполнена при поддержке НАНУ (проекты № 0116U004558 и № 0116U006078) и Volkswagen Foundation (грант № 90 306).

Список литературы

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. Octave [Электронный ресурс] http://www.octave.org. – Режим доступа: свободный.

4. **Шор Н.З., Стеценко С.И**. Квадратичные экстремальные задачи и недифференцируемая оптимизация. Киев: Наук. думка, 1989. 208 с.

5. **Журбенко Н.Г., Марчук Т.В.** Алгоритм минимизации негладких функционалов / $r(\alpha)$-алгоритм/. АН УССР, РФАП, 22, 1976 г.

6. **Шор Н.З., Стецюк П.И.** Использование модификации r-алгоритма для нахождения глобального минимума полиномиальных функций // Кибернетика и систем. анализ. 1997. 4. С. 28–49.

7. **Kappel F., Kuntsevich A.V.** An implementation of Shor's r-algorithm // Computational Optimization and Applications. 2000. Vol. 15, 2. P. 193–205.

8. Математические и программные средства моделирования и оптимизации динамической загрузки мощностей энергосистемы / П.И. Стецюк, А.П. Лиховид, Б.М. Чумаков, А.Ю. Видил, А.В. Пилиповский // Отчет о научно-исследовательской работе № гос. регистрации 0107U004963. – Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украини, 2009. – 136 с. http://www.incyb.kiev.ua/file/d120-energy/Ot2009-2mb.pdf

9. **Стецюк П.И.** Методы эллипсоидов и r-алгоритмы. – Кишинэу, Эврика, 2014. – 488 с.