Субградиентные методы ralgb5 и ralgb4 для минимизации овражных выпуклых функций

П. И. Стецюк

Институт кибернетики им. В.М. Глушкова НАН Украины, Киев, Украина

Контактный e-mail: stetsyukp@gmail.com

Рассмотрены свойства трех вычислительных форм r-алгоритмов Н. З. Шора для оптимизации негладких функций, которые отличаются трудоемкостью одной итерации. Обсуждается вариант r-алгоритмов с адаптивным способом регулировки шага в направлении нормированного антисубградиента в преобразованном пространстве переменных. Описаны Octave-функции `ralgb5` и `ralgb4`, которые реализуют две вычислительно устойчивые формы r-алгоритмов с адаптивной регулировкой шага и постоянным коэффициентом растяжения пространства. Приведены результаты вычислительных экспериментов для существенно овражной кусочно-квадратичной функции и кусочно-линейной функции, которая связана с разрешимостью интервальной линейной задачи о допусках.

Ключевые слова: субградиентный метод, растяжение пространства, r-алгоритм, адаптивная регулировка шага, GNU Octave, Octave-функция, кусочно-квадратичная функция `maxquad`, интервальная линейная задача о допусках.

Введение

Непрерывные функции с разрывным градиентом часто встречаются в различных задачах интервального анализа как следствие негладкости результатов интервальных арифметических операций. Поэтому владение численными методами оптимизации негладких функций дает разработчику действенный инструмент при построении эффективных алгоритмов для решения задач математического программирования в интервальном анализе. Большую помощь здесь могут оказать субградиентные методы с преобразованием пространства для минимизации выпуклых функций.

Негладкие выпуклые функции, как правило, характеризуются овражной или существенно овражной структурой поверхностей уровня. Качество методов минимизации для овражных выпуклых функций главным образом определяется способностью избежать зигзагообразной траектории итерационного процесса в точках, близких ко дну "русла оврага". Если метод с этим легко справляется то, как правило, этот метод обладает ускоренной сходимостью при минимизации негладких выпуклых функций.

В настоящей статье обсуждаются две программные реализации алгоритмов из семейства субградиентных методов с растяжением пространства, которое известно как r-алгоритмы Н. З. Шора [1, с. 90–102], [2, с. 100–112], оно имеет ускоренную сходимость при минимизации выпуклых функций с овражной структурой поверхностей уровня.

Субградиентные методы, в которых используется растяжение пространства в направлении разности двух последовательных субградиентов, предложены в работах [3, 4]. 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 конкурентоспособны как по надежности, так и по времени счета и точности результатов по сравнению 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 субградиента. В результате этого растяжения уменьшаются поперечные составляющие субградиентов вдоль направления на точку минимума, что обеспечивает более быструю сходимость субградиентного процесса с растяжением пространства. Вторая идея позволяет улучшить свойства овражной функции в преобразованном пространстве переменных.

Комбинации этих двух идей, которые определяются регулировкой шага наискорейшего спуска (точного или приближенного) и соответствующим выбором коэффициента растяжения пространства, обеспечивают ускоренную сходимость конкретных вариантов r-алгоритмов и гарантируют их монотонность (или почти монотонность) по значению минимизируемой функции. Это подтверждается результатами многочисленных тестовых и реальных расчетов в задачах линейного и нелинейного программирования, блочных задачах с различными схемами декомпозиции, при решении минимаксных и матричных задач оптимизации, для вычисления двойственных лагранжевых оценок в многоэкстремальных и комбинаторных задачах оптимизации [1, 2].

В статье описаны субградиентные методы `ralgb5` и `ralgb4`, которые являются реализацией двух форм r-алгоритмов с адаптивной регулировкой шага в направлении нормированного антисубградиента в преобразованном пространстве переменных и постоянным коэффициентом растяжения пространства. Первый метод `ralgb5` реализован одноименной функцией на языке Octave [5] и базируется на так называемой устойчивой B-форме r-алгоритмов, где B означает неособенную $n\times n$-матрицу. Его название связано с тем, что коррекция матрицы B на каждой итерации метода требует $5n^2$ арифметических операций умножения. Второй метод реализован Octave-функцией `ralgb4` и базируется на менее устойчивой, но экономной B-форме r-алгоритмов, в которой на итерации требуется $4n^2$ арифметических операций умножения.

Статья содержит пять разделов. В первом разделе описаны три вычислительные формы r-алгоритмов и проанализированы их свойства. Во втором разделе описаны $r(\alpha)$-алгоритм с адаптивной регулировкой шага и имеющиеся его программные реализации. В третьем разделе приведены Octave-функции `ralgb5` и `ralgb4` и их коды. В четвертом разделе описаны результаты вычислительных экспериментов для существенно овражной кусочно-квадратичной функции `maxquad` и кусочно-линейной функции, ко-

торая связана с разрешимостью интервальной линейной задачи о допусках, на примере интервальной линейной системы с матрицей Ноймайера.

1. О трех вычислительных формах *r*-алгоритмов

Пусть $f(x)$ — выпуклая функция аргумента x, который является вектором из n вещественных переменных, т. е. $x \in \mathbb{R}^n$. Минимальное значение функции $f(x)$ на $\mathbb{R}^n$ будем обозначать посредством $f^* = f(x^*)$, где x^* — аргумент точки минимума. Мы предполагаем впредь, что $f(x)$ имеет ограниченное множество минимумов X^*, т. е. выполняется условие

$$\lim_{\|x\|\to\infty} f(x) = +\infty,$$

где $\|\cdot\|$ — некоторая норма в $\mathbb{R}^n$. Это условие обеспечивает корректность процедуры регулировки шага в r-алгоритмах. Наконец, пусть $\{\alpha_k\}_{k=0}^{\infty}$ обозначает набор вещественных чисел, удовлетворяющих условию $\alpha_k > 1$, которые будут служить коэффициентами растяжения пространства.

r-Алгоритмом для минимизации функции $f(x)$ 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:

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

где

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

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

Здесь x_0 — стартовая точка; $B_0 = I_n$ — единичная $n\times n$-матрица[1]; h_k^* — величина шага, выбираемая 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 $\beta = \alpha^{-1} < 1$; $g_f(x_k)$ 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

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

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

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

функции f в точке x_k удовлетворяет неравенству

$$f(x) \geq f(x_k) + \big(g_f(x_k)\big)^T (x - x_k)$$

для любого $x \in \mathbb{R}^n$. Отсюда, осуществляя замену переменных $x = B_k y$, получаем

$$\varphi(y) \;\geq\; \varphi(y_k) + \Big(B_k^T g_f(x_k)\Big)^T (y - y_k) \;=\; \varphi(y_k) + \Big(g_\varphi(y_k)\Big)^T (y - y_k)$$

для любого $y \in \mathbb{R}^n$. 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 в условии останова r-алгоритмов для негладких функций.

Метод (1)–(3) называют *B-формой r-алгоритмов*: на каждой его итерации корректируется матрица, связанная с заменой переменных $x = By$. Одна 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 общую 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 n^2 умножений требуется для вычисления вектора $\eta = B_k\eta_k$ и столько же умножений требует построение одноранговой матрицы $\eta\,\eta_k^T$.

У r-алгоритмов имеется еще одна разновидность B-формы, которая по сравнению с методом (1)–(3) позволяет сэкономить n^2 операций умножения на каждой итерации. Этот экономный r-алгоритм является итеративной процедурой нахождения последовательностей векторов $\{x_k\}_{k=0}^{\infty}$ и матриц $\{B_k\}_{k=0}^{\infty}$ по следующему правилу:

$$x_{k+1} = x_k - h_k B_k \frac{\tilde{g}_k}{\|\tilde{g}_k\|}, \quad B_{k+1} = B_k R_{\beta_k}(\eta_k), \quad \tilde{g}_{k+1} = R_{\beta_k}(\eta_k) g_{k+1}^*, \quad k = 0, 1, 2, \ldots, \quad (5)$$

где

$$h_k \geq h_k^* = \arg\min_{h \geq 0} f\left(x_k - h B_k \frac{\tilde{g}_k}{\|\tilde{g}_k\|}\right), \quad g_{k+1}^* = B_k^T g_f(x_{k+1}), \quad (6)$$

$$\eta_k = \frac{g_{k+1}^* - \tilde{g}_k}{\|g_{k+1}^* - \tilde{g}_k\|}, \quad \beta_k = \frac{1}{\alpha_k}. \quad (7)$$

Здесь x_0 — стартовая точка, такая что $x_0 \neq x^*$; $\tilde{g}_0 = B_0^T g_f(x_0)$, где B_0 — неособенная $n \times n$-матрица; $g_f(x_k)$ и $g_f(x_{k+1})$ — субградиенты функции $f(x)$ в точках x_k и x_{k+1}. Если $g_f(x_{k+1}) = 0$, то $x_{k+1} = x^*$ и процесс (5)–(7) останавливается.

Итерация метода (5)–(7) требует порядка $4n^2$ операций умножения. Из них $2n^2$ умножений необходимы для вычисления векторов $B_k\tilde{g}_k$ и $B_k^T g_f(x_{k+1})$, а $2n^2$ умножений требует одноранговый пересчет матрицы $B_{k+1} = B_k R_{\beta_k}(\eta_k)$. Экономия в n^2 умножений связана с тем, что $\tilde{g}_{k+1} = B_{k+1}^T g_f(x_{k+1})$ — субградиент в пространстве переменных

$y = A_{k+1}x$ пересчитывается с учетом уже вычисленного $g^*_{k+1} = B_k^T g_f(x_{k+1})$ — субградиента в пространстве переменных $y = A_k x$. Пересчет субградиента $\tilde{g}_{k+1}$ выполняется по формуле

$$\begin{aligned}\tilde{g}_{k+1} &= B_{k+1}^T g_f(x_{k+1}) = R_{\beta_k}(\eta_k) g^*_{k+1} = \\ &= (I_n + (\beta_k - 1)\eta_k \eta_k^T) g^*_{k+1} = g^*_{k+1} + (\beta_k - 1)(\eta_k^T g^*_{k+1})\eta_k,\end{aligned}$$

которая не требует операции умножения матрицы на вектор. Этот пересчет способствует большему накоплению ошибок при вычислении нормированного субградиента в преобразованном пространстве по формуле $\xi_k = \dfrac{\tilde{g}_k}{\|\tilde{g}_k\|}$, чем при вычислении нормированного субградиента по формуле $\xi_k = \dfrac{B_k^T g_f(x_k)}{\|B_k^T g_f(x_k)\|}$, как в методе (1)–(3).

Помимо рассмотренных выше форм r-алгоритмы можно записать еще в так называемой *H-форме* (по типу методов переменной метрики), введя в рассмотрение симметричную матрицу $H_k = B_k B_k^T$. Тогда r-алгоритмы в H-форме задаются итеративной процедурой нахождения последовательностей векторов $\{x_k\}_{k=0}^{\infty}$ и симметричных матриц $\{H_k\}_{k=0}^{\infty}$ по следующему правилу:

$$x_{k+1} = x_k - h_k \frac{H_k g_f(x_k)}{\sqrt{g_f^T(x_k) H_k g_f(x_k)}}, \quad H_{k+1} = H_k + (\beta_k^2 - 1)\frac{H_k r_k r_k^T H_k}{r_k^T H_k r_k}, \quad k = 0, 1, 2, \ldots, \quad (8)$$

где

$$h_k \geq h_k^* = \arg\min_{h \geq 0} f\left(x_k - h \frac{H_k g_f(x_k)}{\sqrt{g_f^T(x_k) H_k g_f(x_k)}} \right), \quad (9)$$

$$\beta_k = \frac{1}{\alpha_k} < 1, \quad r_k = g_f(x_{k+1}) - g_f(x_k). \quad (10)$$

Здесь x_0 — стартовая точка; $H_0 = I_n$ — единичная $n \times n$-матрица; h_k — величина шага, которая не меньше чем h_k^*; $g_f(x_k)$ 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 (8)–(10) останавливается.

H-форма r-алгоритмов более экономна, чем B-форма: по памяти — почти в два раза, так как требуется хранить симметричную матрицу, а по трудоемкости — как минимум в 5/3 раз. Действительно, если даже симметричную матрицу H_k хранить как полную матрицу размером $n \times n$, то итерация метода (9), (10) требует порядка $3n^2$ операций умножения: $2n^2$ умножений требуется для вычисления векторов $H_k g_f(x_k)$ и $\eta = H_k r_k$, а n^2 умножений — для вычисления одноранговой матрицы $\eta\eta^T = H_k r_k r_k^T H_k$, которая используется при пересчете матрицы H_{k+1}. Но этот выигрыш нивелируется тем, что H-форма r-алгоритмов вычислительно менее устойчива, чем B-форма, и при ее реализации требуется учитывать дополнительные условия. Так, например, для метода (9), (10) необходимо контролировать, чтобы матрица H_k была положительно определенной. В методах (1)–(3) и (5)–(7) такого контроля не требуется, так как вычисления, хотя и неявно, связаны с положительно определенной матрицей $H_k = B_k B_k^T$.

Вычислительные характеристики описанных форм r-алгоритмов по памяти и трудоемкости подытожены в табл. 1. И хотя теоретически все три формы r-алгоритмов

Т а б л и ц а 1. Вычислительные характеристики трех форм r-алгоритмов

Форма r-алгоритмов	Вид метода	Оперативная память	Трудоемкость итерации	Устойчивость метода
B-форма	Метод (1)–(3)	$\sim n^2$	$\sim 5n^2$	хорошая(+)
Экономная B-форма	Метод (5)–(7)	$\sim n^2$	$\sim 4n^2$	хорошая
H-форма	Метод (8)–(10)	$\sim n^2/2$	$\sim 3n^2$	средняя

одинаковы, вычислительная устойчивость их компьютерных реализаций, которая представлена в последнем столбце, будет различной. Для обеих B-форм она отмечена как "хорошая", но преимущество отдано методу (1)–(3) и отмечено знаком "+". Это обусловлено тем, что при переходе в очередное пространство переменных пересчет субградиента, который используется в экономной B-форме, способствует накоплению ошибок по отношению к вычислению этого же субградиента в методе (1)–(3). Вычислительная устойчивость r-алгоритмов в H-форме отмечена как "средняя". Это означает, что с их помощью нельзя найти такие наилучшие приближения к точке минимума, которые можно найти с помощью r-алгоритмов в B-форме. Ситуация здесь близка к той, которая имеет место при компьютерном вычислении величины $t = 1 - \varepsilon^2$ для очень малых величин ε при ограниченной разрядности арифметических операций. Так, если величину t вычислять по указанной формуле, то требования к разрядности компьютерной арифметики будут в два раза более сильными, чем если ее вычислять по формуле $t = (1 - \varepsilon)(1 + \varepsilon)$. Поэтому сохранить вычислительную устойчивость r-алгоритмов в H-форме такой же, как и для r-алгоритмов, в B-форме невозможно. Их можно использовать, если не требуется высокой точности к нахождению минимума функции $f(x)$.

2. Об $r(\alpha)$-алгоритме с адаптивной регулировкой шага

Проблема обоснования сходимости r-алгоритмов для всего класса выпуклых функций все еще остается открытой. Одна из причин, по которой сложно построить гарантированно сходящийся вариант r-алгоритмов для негладких функций, связана с неоднозначным выбором антисубградиента для последующего направления движения из точек негладкости, где функция недифференцируема, а ее субградиенты линейно зависимы и ни один из антисубградиентов не является направлением убывания функции. В работе [6] показано, что для выпуклой кусочно-линейной функции такие точки негладкости могут служить ловушками для минимизирующей последовательности варианта r-алгоритмов с $\alpha_k = \alpha = \text{const}$ и $h_k = h_k^*$. В [7] построена модификация r-алгоритмов, которая для кусочно-линейной функции от двух переменных позволяет преодолеть точки-ловушки одного конкретного вида. Реализуется это за счет хранения некоторого количества субградиентов и выбора в качестве второго субградиента того, который с первым субградиентом образует наибольший угол в преобразованном пространстве переменных.

Применительно к задачам минимизации гладких функций r-алгоритмы по своей формальной структуре близки к алгоритмам квазиньютоновского типа с переменной метрикой. Так, предельный вариант r-алгоритма с бесконечным коэффициентом растяжения (здесь $\beta_k = \beta = 0$, $h_k = h_k^*$) является проективным вариантом метода сопряженных градиентов [2, с. 101–102]. Для задачи минимизации выпуклой непрерывно

дифференцируемой функции $f(x)$ предельный вариант r-алгоритма с восстановлением матрицы B_k после каждых n итераций обладает квадратичной скоростью сходимости при обычных условиях гладкости и регулярности $f(x)$ (теорема 3.10, см. [1, с. 94–98]). Этим в значительной мере объясняется замечательное свойство r-алгоритмов, которое заключается в том, что их конкретные реализации показывают очень хорошие результаты при минимизации овражных выпуклых функций.

Одним из самых эффективных зарекомендовал себя $r(\alpha)$-алгоритм с адаптивной регулировкой шага, для которого коэффициент растяжения пространства выбирается одним и тем же на каждой итерации. Это вариант r-алгоритмов, в котором $\alpha_k = \alpha > 1$ и 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 ($q_1 \leq 1$), если условие завершения спуска по направлению выполняется за один шаг; q_2 — коэффициент увеличения шага ($q_2 \geq 1$); n_h — натуральное число, задающее число шагов одномерного спуска ($n_h > 1$), через каждые из которых шаг будет увеличиваться в q_2 раз.

Условием завершения спуска по направлению является обнаружение точки x_{k+1}, для которой $(x_{k+1} - x_k)^T g_f(x_{k+1}) \geq 0$. Оно 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 $\lim\limits_{\|x\| \to \infty} f(x) = +\infty$, 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 $f(x)$ не ограничена 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 [8, с. 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: $\alpha = 2 \dots 4$, $h_0 = 1.0$, $q_1 = 1.0$, $q_2 = 1.1 \dots 1.2$, $n_h = 2 \dots 3$. 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 ($q_1 = 0.8 \dots 0.95$). Это обусловлено тем, что дополнительное измельчение шага способствует увеличению точности поиска минимума функции по направлению, которое 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 ε_x, $\varepsilon_g \sim 10^{-6} \dots 10^{-5}$ при минимизации выпуклой функции, даже существенно овражной структуры, обеспечивает нахождение точки x_r^* — приближения к точке x^*, для которого значение функции достаточно близко к оптимальному: для негладких функций

$$\frac{f(x_r^*) - f(x^*)}{|f(x^*)| + 1} \sim 10^{-6} \dots 10^{-5};$$

для гладких функций

$$\frac{f(x_r^*) - f(x^*)}{|f(x^*)| + 1} \sim 10^{-12} \dots 10^{-10}.$$

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. В их основе лежат B-формы r-алгоритмов. Экономная B-форма, которая использует метод (5), (6) или незначительные его модификации, дала 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#). Компьютерные программы для r-алгоритмов в H-форме практически забыты, так как H-форма считается неэффективной для достаточно точного решения задач оптимизации. Ее используют для решения задач с простыми ограничениями (линейные равенства и двусторонние границы на переменные), для которых H-форма r-алгоритмов предоставляет ряд удобств при реализации метода проекции субградиента, которые не предоставляет B-форма.

Исторически одной из первых была фортрановская программа `ralg` (автор Н.Г. Журбенко) для решения вычислительных задач на компьютерах БЭСМ-6 и ЕС-1040 [9]. Поскольку в программе основной массив необходимой памяти занимает матрица B, для хранения которой необходимо n^2 ячеек, то, используя лишь оперативную память ЕС-1040 (640 К), можно было решать задачи с количеством переменных до 250. Поэтому параметры адаптивной регулировки шага и их связь с коэффициентом растяжения пространства проверялись на задачах с несколькими сотнями переменных. В 70–80 гг. прошлого века программа `ralg` активно использовалась сотрудниками и аспирантами как Института кибернетики АН Украины, так и других организаций, которые сталкивались с задачами оптимизации негладких функций. Так, например, активными пользователями программы были Д.И. Соломон (г. Кишинев, Институт математики) и Е.М. Киселева (Днепропетровский университет), которые впоследствии защитили докторские диссертации под руководством Н.З. Шора.

В 1990-е годы программа `ralg` послужила основой для программной реализации фортрановской программы `ralgb4` (автор П.И. Стецюк), которая использует модификацию r-алгоритма из [10]. С помощью программы `ralg` А.В. Кунцевич разработал комплекс программ `SolveOpt` (языки Фортран и Си), где использованы усложненная адаптивная регулировка шага и критерии останова $|x^i_{k+1} - x^i_k| \le \delta_x |x^i_{k+1}|$ и $|f(x_{k+1}) - f(x_k)| \le \delta_f |f(x_{k+1})|$ с заданными достаточно малыми δ_x и δ_f [11]. 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. Потери в n^2 арифметических операций умножения по сравнению с экономной B-формой оказались малозаметными, так как трудоемкость вычисления значения функции и субградиента в точке часто значительно превосходила эти потери. Но зато алгоритмическая реализация $r(\alpha)$-алгоритма с адаптивной регулировкой шага упростилась и она стала вычислительно устойчивой для достаточно точного нахождения приближений к точке минумума. Фортрановский код программы `ralgb5` приведен на с. 34–38 отчета [12], 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 г. программа `ralgb5` с помощью Е.А. Нурминского (Владивосток, ИАПУ ДВО РАН) была переписана на язык Octave. Код Octave-программы `ralgb5` приведен в [13, с. 384–385]. Она оказалась быстрее фортрановской программы, если речь шла о решении задач для тысячи и более переменных. Это 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 2011 г. Octave-программа переписана С.П. Шарым (Новосибирск, ИВТ СО РАН) в программу на языке Scilab'а [14]. Оказалось, что она более устойчива к нахождению решений, чем аналогичный код, который был создан ранее на основе фортрановской программы `ralgb4`. В настоящее время программа `ralgb5` переведена А.П. Лиховидом на языки C++, C# и используется в программных реализациях алгоритмов решения различного рода задач нелинейного программирования.

3. Octave-функции ralgb5 и ralgb4

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 выбором следующих параметров:

- 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 заменено любым другим допускаемым синтаксисом 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`:

- `2` — 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);
- `3` — 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);
- `4` — 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);
- `5` — 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 $f(x)$ не ограничена 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 выбраны из тех соображений, что для рекомендуемого минимального значения $q_2 = 1.1$ начальный шаг для очередной итерации мог бы 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 выглядит следующим образом:

```
# 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 бóльшая часть из них содержит два и более оператора языка Octave. Функционально он полностью совпадает с кодом программы `ralgb5`, который приведен в [13, с. 384–385], и отличается от него только перенумерацией 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-й итерации субградиент $g_f(x_k)$ запоминается вектор-столбцом `g0`, а субградиент $g_f(x_{k+1})$ — 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 субградиента `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` позволяют 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.

При разработке оптимизационных ядер важная роль отводится усовершенствованиям программы `ralgb5`, направленным на ускорение итерационного процесса для получения решения задачи с заданной точностью. Так, например, если оптимальные переменные в задаче имеют разный масштаб, то для останова процесса целесообразно использовать одно из условий:

$$\sqrt{\sum_{i=1}^{n}\left(\frac{|x_{k+1}^i - x_k^i|}{|x_k^i|+1}\right)^2} \le \varepsilon_x \quad \text{или} \quad \max_{i=1,\ldots,n}\frac{|x_{k+1}^i - x_k^i|}{|x_k^i|+1} \le \delta_x.$$

Они лучше, чем условие $\|x_{k+1} - x_k\| \le \varepsilon_x$, учитывают большие и малые значения переменных, и их использование при малых ε_x или δ_x будет адекватно отражать останов для разномасштабных переменных и не изменять его, если переменные одного порядка. Более быстрой остановке итерационного процесса будут способствовать условия останова по отклонениям значений минимизируемой функции от ее рекордного значения f_r. Так, например, если на последних $m = n$ итерациях f_r ни разу не улучшалось, то с большой вероятностью мы находимся в достаточно малой окрестности точки минимума. С помощью указанных условий останова метод `ralgb5` реалистично ускорить в два — три раза, а если их сочетать с настройкой параметров адаптивного шага на конкретное семейство оптимизационных задач, то ускорение метода может быть и более существенным.

Программу `ralgb5` легко превратить в программу `ralgb4`, которая будет реализовывать экономную B-форму r-алгоритмов по формулам метода (5), (6). Для этого достаточно в коде функции `ralgb5` заменить четыре строки (7, 21–23) на следующие:

```
dx = B*g0/norm(g0);                                           # row007b
g1 = B'*g1; xi = g1 - g0; xi = xi /norm(xi);                  # row021b
B += (1 / alpha - 1) * B * xi * xi';                          # row022b
g0 = g1 + (1 / alpha - 1) * xi * (xi'*g1);                    # row023b
```

Недостатком этой замены является только то, что вектор `g1` переприсваивается в строке `21b` с помощью оператора `g1 = B'*g1`. Чтобы избежать этого переприсвоения, достаточно во внутреннем цикле `while` (строки 9–17) операции с вектором `g1` заменить на операции с вектором `g`, а оператор переприсвоения заменить на оператор `g1 = B'*g`.

Такие минимальные изменения в коде программы `ralgb5` позволяют получить 26-строчный код для Octave-функции `ralgb4`:

```
# Octave-function ralgb4 for Shor's r-algorithm
function [xr,fr,itn,ncalls,istop] = ralgb4(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 %16.8e fr %21.13e 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*g0/norm(g0);                                                 # row007
   d = 1; ls = 0; ddx = 0;                                             # row008
   while (d > 0)                                                       # row009
      x -= hs * dx; ddx += hs * norm(dx);                              # row010
      ncalls ++; [f, g] = calcfg(x);                                   # row011
      if (f < fr)  fr = f; xr = x; endif                               # row012
      if(norm(g) < epsg) istop = 2;  return; endif                     # row013
      ls ++; (mod(ls,nh)==0) && (hs *= q2);                            # row014
      if(ls > 500) istop = 5;  return; endif                           # row015
      d = dx' * g;                                                     # row016
  endwhile                                                             # row017
  (ls == 1) && (hs *= q1);                                             # row018
  printf("itn %4d f %16.8e fr %21.13e ls %2d ncalls %4d\n",            # row019
         itn, f, fr, ls, ncalls);
  if(ddx < epsx) istop = 3;  return; endif                             # row020
  g1=B'*g; xi = g1 - g0; xi = xi /norm(xi);                            # row021
  B += (1 / alpha - 1) * B * xi * xi';                                 # row022
  g0 = g1 + (1 / alpha - 1) * xi * (xi'*g1);                           # row023
endfor                                                                 # row024
istop = 4;                                                             # row025
endfunction                                                            # row026
```

Приведенный код Octave-функции `ralgb4` отвечает итерационному процессу по формулам (5), (6), который выполняется в цикле `for` (строки `6-24`), где для k-й итерации субградиент $g_\varphi(y_k) = B_k^T g_f(x_k)$ запоминается вектор-столбцом `g0`, а субградиент $g_\varphi(y_{k+1}) = B_k^T g_f(x_{k+1})$ — вектор-столбцом `g1`.

С помощью программ `ralgb5` и `ralgb4` можно находить достаточно точные приближения к точке минимума выпуклой функции. При правильном подборе коэффициента растяжения α и параметров адаптивной регулировки шага h_0, q_1, n_h и q_2 можно значительно сократить количество итераций для выполнения одних и тех же критериев останова. Это зависит от конкретного вида минимизируемой функции, степени ее овражности и масштаба переменных.

4. Программы ralgb5 и ralgb4 для задач maxquad и shary

Опишем результаты трех вычислительных экспериментов с программами `ralgb5` и `ralgb4`. Первый и второй эксперименты связаны с достаточно точным и очень точным поисками минимума для известной кусочно-квадратичной функции `maxquad` [15, с. 151]. Третий эксперимент связан с исследованием параметров метода `ralgb4` для максимизации кусочно-линейной функции `shary`, которая позволяет установить разрешимость интервальной линейной задачи о допусках. Вычисления проводились на компьютере Pentium 3GHz в системе Windows7/32 в среде программирования GNU Octave версии 3.6.4.

Задача `maxquad` состоит в минимизации существенно овражной выпуклой негладкой функции от десяти переменных: $f(x) = \max\limits_{1\le k\le 5} f_k(x)$, где $f_k(x) = x^T A_k x - b_k^T x$,

A_k — симметричные 10×10-матрицы, такие что $A_{kij} = e^{i/j}\cos(ij)\sin k$, если $i < j$, и $A_{kii} = i\,|\sin k|\,/10 + \sum\limits_{j\neq i} |A_{kij}|$, а компоненты векторов b_k определяются $b_{ki} = e^{i/k}\sin(ik)$. Начальным приближением является точка $x_0 = (1,\ldots,1)^T \in \mathbb{R}^{10}$, в которой реализуется значение функции $f(x_0) = 5337.06643$. Функция `maxquad` имеет единственную точку минимума, минимальное значение функции f^* определяется его достаточно точным приближением $f_{\min} = -0.841408334596$ (с точностью 10^{-12}, т. е. 12 цифр после десятичной точки).

Первый эксперимент состоит в проверке устойчивости метода `ralgb5` по отношению к достаточно точному поиску минимума функции `maxquad`. Он связан с решением с помощью программы `ralgb5` восемнадцати задач `maxquad`, где каждая отдельная задача определяется возможной комбинацией одного из трех значений коэффициента растяжения $\alpha \in \{2,3,4\}$ и одного из шести значений параметра останова $\varepsilon_x \in \{10^{-10}, 10^{-9}, \ldots, 10^{-5}\}$. Решение указанных восемнадцати задач реализует следующий Octave-код:

```
global B b m;
n = 10; m = 5; B = zeros(n, n, m); b = zeros(m, n);
en = [ 1:n ];  a1 = en'*ones(1,n);
a2 = exp(min(a1, a1') ./ max(a1, a1')); a3 = cos(en'*en);
for k = 1:m
   A = a2 .* a3 * sin(k); A = A - diag(diag(A));
   B(:,:,k) = A + diag(sum(abs(A)) + abs(sin(k))*en/n);
   b(k, :) = exp(en/k) .* sin(en*k);
endfor
fmin = -0.841408334596, x0 = ones(n,1),
 alpha = [2.0 3.0 4.0], h0 = 1.0, nh = 3, q1 = 1.0, q2 = 1.1
#alpha = [2.0 3.0 4.0], h0 = 1.0, nh = 3, q1 = 0.8, q2 = 1.1
epsx = 1.e-5, epsg = 1.e-6, maxitn = 1000
epsx1 = [ ]; itn = [ ]; nfg = [ ];  df = [ ];
for i=1:6
  itn1 = [ ]; nfg1 = [ ]; df1 = [ ];
  for j=1:3
    alp = alpha(j),  x = x0;
    [xr,fr,itr,ncalls,istop] = ralgb5(@maxquad,x,alp,h0,q1,q2,nh,epsg,epsx,maxitn),
    itn1 = [itn1 itr]; nfg1 = [nfg1 ncalls]; df1 = [df1 fr-fmin];
  endfor
  itn = [itn; itn1]; nfg = [nfg; nfg1]; df = [df; df1];
  epsx1 = [epsx1; epsx]; epsx = epsx/10.d0,
endfor
printf("              alpha = %3.1f         alpha = %3.1f",alpha(1),alpha(2));
printf("         alpha = %3.1f \n",alpha(3));
printf("  ..epsx.. itn(nfg)  fr-fmin   itn(nfg)  fr-fmin   itn(nfg)  fr-fmin\n");
for i=1:6
   printf("%9.1e ",epsx1(i,1));
   for j=1:3
      printf(" %3d(%3d) %9.1e ",itn(i,j),nfg(i,j),df(i,j));
   endfor
   printf("\n");
endfor
```

в котором вычисление $f(x)$ и $g_f(x)$ выполняется с помощью Octave-функции

```
function [f,g] = maxquad(x)
global B b m;
ff = zeros(1,m);
for k = 1:m
   ff(k) =  x'*B(:,:,k)*x - b(k, :)*x;
endfor
[ f indx ] = max(ff);
g = 2*x'*B(:,:,indx) - b(indx,:);
g = g';
endfunction
```

Результатом работы этого кода являются подробный протокол работы метода `ralgb5` по решению задач `maxquad` для каждой из возможных пар (α,ε_x) и итоговая таблица, в которой для всех восемнадцати задач приведены затраты на их решение по количеству итераций `itn`, количеству вычислений значения функции и ее субградиента `nfg` и отклонение найденного рекордного значения f_r от оптимального $f_{\min} = -0.841408334596$. Главная программа содержит одну закомментированную строку, которая помещена в код для удобства расчета итоговой таблицы при двух разных значениях коэффициента уменьшения шага $q_1 \in \{1.0, 0.8\}$. Если $q_1 = 1.0$, то итоговая таблица имеет такой вид:

	alpha = 2.0		alpha = 3.0		alpha = 4.0	
..epsx..	itn(nfg)	fr-fmin	itn(nfg)	fr-fmin	itn(nfg)	fr-fmin
1.0e-005	148(164)	4.8e-007	90(124)	1.7e-006	87(132)	2.6e-007
1.0e-006	175(195)	3.1e-008	107(144)	1.0e-007	102(153)	2.0e-008
1.0e-007	211(236)	5.9e-010	133(179)	7.3e-010	114(174)	1.2e-009
1.0e-008	240(267)	3.9e-011	159(211)	2.3e-011	141(218)	5.5e-012
1.0e-009	278(309)	1.7e-013	185(247)	4.0e-014	154(237)	2.7e-013
1.0e-010	330(368)	-4.1e-013	223(294)	-4.1e-013	180(274)	-4.1e-013

а если $q_1 = 0.8$, то итоговая таблица будет такой:

	alpha = 2.0		alpha = 3.0		alpha = 4.0	
..epsx..	itn(nfg)	fr-fmin	itn(nfg)	fr-fmin	itn(nfg)	fr-fmin
1.0e-005	68(114)	1.3e-007	73(156)	1.0e-007	63(153)	3.3e-007
1.0e-006	71(120)	3.7e-008	85(180)	4.0e-009	75(175)	9.2e-009
1.0e-007	80(135)	3.6e-009	95(200)	3.3e-010	75(175)	9.2e-009
1.0e-008	102(167)	8.2e-012	104(217)	2.7e-011	96(219)	3.4e-012
1.0e-009	105(170)	1.8e-012	118(241)	1.1e-013	106(236)	-1.5e-013
1.0e-010	110(176)	-3.2e-013	127(257)	-3.6e-013	114(253)	-4.0e-013

Из обеих итоговых таблиц видно, что последовательное уменьшение ε_x в 10 раз не приводит к резкому увеличению затрат по количеству итераций, что свидетельствует об устойчивости метода `ralgb5` по отношению к достаточно точному поиску минимума. Если $\varepsilon_x = 10^{-10}$, то при всех коэффициентах растяжения $\alpha \in \{2, 3, 4\}$ и значениях параметра $q_1 \in \{1.0, 0.8\}$ минимум функции `maxquad` найден с точностью до всех двенадцати цифр после точки, о чем сигнализируют отрицательные значения $f_r - f_{\min}$. Если $q_1 = 1.0$, то наименьшие затраты составляют 180 `itn` и 274 `nfg` и реализуются при коэффициенте растяжения $\alpha = 4$, а если $q_1 = 0.8$, то наименьшие затраты составляют 110 `itn` и 176 `nfg` и реализуются при коэффициенте растяжения $\alpha = 2$.

Для нахождения минимума функции maxquad с точностью $1.3 \cdot 10^{-7}$ наименьшие затраты метода ralgb5 реализуются при $\alpha = 2$ и $q_1 = 0.8$ и составляют 68 itn и 114 nfg. Для сравнения заметим, что алгоритму 16.2 [16, с. 241] для достижения почти такой же точности $2 \cdot 10^{-7}$ потребовалось 97 итераций и 282 вычисления значения функции и ее субградиента (см. табл. 2 в [16, с. 282]). Однако при выборе параметра $q_1 = 0.8$ следует соблюдать осторожность, так как из-за сильного уменьшения величины пробного шага по направлению этот выбор может привести к преждевременному срабатыванию критерия останова для негладких функций. В данной ситуации этого нет, так как отношение nfg/itn=176/110 и, следовательно, уменьшение величины шага происходит не на каждой итерации метода ralgb5.

Второй эксперимент для задачи maxquad состоит в уточнении значения $f_{\min}$ с 12 до 15 значащих цифр. Реализуется это с помощью методов ralgb5 и ralgb4, которые при $\varepsilon_x = 10^{-11}$ находят решения десяти задач maxquad, различающихся начальными стартовыми точками. Первая задача использует стандартную начальную стартовую точку, а для остальных девяти задач стартовая точка выбирается в гиперкубе $[-1, 1]^{10}$ с помощью Octave-датчика rand, который реализует равномерное распределение случайной величины на отрезке $[0, 1]$. Для методов ralgb5 и ralgb4 выбраны параметры: $\alpha = 2$, $h0 = 1$, $q_1 = 1.0$, $n_h = 3$, $q_2 = 1.1$.

Второй вычислительный эксперимент реализован таким Octave-кодом:

```
global B b m;
n = 10; m = 5; B = zeros(n, n, m); b = zeros(m, n);
en = [ 1:n ];  a1 = en'*ones(1,n);
a2 = exp(min(a1, a1') ./ max(a1, a1')); a3 = cos(en'*en);
for k = 1:m
   A = a2 .* a3 * sin(k); A = A - diag(diag(A));
   B(:,:,k) = A + diag(sum(abs(A)) + abs(sin(k))*en/n);
   b(k, :) = exp(en/k) .* sin(en*k);
endfor
alpha = 2.0, h0 = 1.0, nh = 3, q1 = 1.0, q2 = 1.1
epsx = 1.e-11, epsg = 1.e-6, maxitn = 1000
fx0 = []; itn = [ ]; nfg = [ ];  fr = [ ]; fr1min=0.0; fr2min=0.0;
x0=ones(n,1); ntest=10; rand("seed", 2016);
for i=1:ntest
   x = x0; [f0 g0]= maxquad(x0); fx0 = [fx0; f0];
   [xr,fr1,itn1,nfg1,istop] = ralgb5(@maxquad,x,alpha,h0,q1,q2,nh,epsg,epsx,maxitn),
   x = x0; if(fr1<=fr1min) fr1min=fr1; xr1min=xr; endif
   [xr,fr2,itn2,nfg2,istop] = ralgb4(@maxquad,x,alpha,h0,q1,q2,nh,epsg,epsx,maxitn),
   if(fr2<=fr2min) fr2min=fr2; xr2min=xr; endif
   itn = [itn; itn1 itn2]; nfg = [nfg; nfg1 nfg2]; fr = [fr; fr1 fr2];
   x0 = 2.*(rand(10,1)-0.5*ones(10,1));
endfor
printf(" .f(x0).  itn(nfg)          fr          itn(nfg)          fr \n");
for i=1:ntest
   printf("%8.2f ",fx0(i,1));
   for j=1:2
      printf(" %3d(%3d) %15.16f ",itn(i,j),nfg(i,j),fr(i,j));
   endfor
   printf("\n");
endfor
```

Его выходом являются подробный протокол работы методов `ralgb5` и `ralgb4` по решению всех десяти задач `maxquad` и итоговая таблица:

```
 .f(x0).  itn(nfg)          fr          itn(nfg)          fr
5337.07  367(415) -0.8414083345964150  366(413) -0.8414083345964152
  82.82  365(403) -0.8414083345964148  364(402) -0.8414083345964147
 133.96  359(400) -0.8414083345964150  360(407) -0.8414083345964147
  87.65  363(404) -0.8414083345964150  351(392) -0.8414083345964147
9405.93  360(406) -0.8414083345964150  385(446) -0.8414083345964150
  91.66  370(416) -0.8414083345964150  359(404) -0.8414083345964150
7844.94  381(450) -0.8414083345964150  378(439) -0.8414083345964150
 152.13  378(420) -0.8414083345964150  378(425) -0.8414083345964150
 107.75  376(431) -0.8414083345964152  364(410) -0.8414083345964150
5653.48  404(493) -0.8414083345964150  364(409) -0.8414083345964150
```

Здесь для каждой из десяти задач `maxquad` приведены значения функции в стартовой точке (первая колонка), затраты `itn(nfg)` и найденное значение `fr` для метода `ralgb5` (колонки 2 и 3) и метода `ralgb4` (колонки 4 и 5).

Из таблицы видно, что для функции `maxquad` минимальное значение f^* можно заменить его 15-значным приближением $f_{\min} = -0.841408334596415$, которое является более точным, чем 12-значное приближение $f_{\min} = -0.841408334596$. При этом затраты на его нахождение методами `ralgb5` и `ralgb4` близки с незначительным превосходством метода `ralgb4`. Но, если у метода `ralgb5` всего один раз из десяти (задача 2) значение f_r чуть-чуть не дотягивает до 15-значного приближения $f_{\min}$, то у метода `ralgb4` это случается трижды (задачи 2, 3 и 4). Минимальное значение f_r достигается каждым из методов по одному разу: для метода `ralgb5` оно достигается для девятой задачи, а для метода `ralgb4` — для первой задачи, которая использует стандартную начальную стартовую точку. Заметим, что для модификации r-алгоритма [10], использующей немного измененную экономную B-форму r-алгоритмов, найти такие значения f_r мешает контроль малости нормы вектора для последующего растяжения пространства.

Octave-коды первого и второго экспериментов для задачи `maxquad` можно проверить на любом из компьютеров, где установлена GNU Octave. Результаты расчета могут несущественно отличаться от приведенных выше. Это связано с тем, что задачи требуется решать с очень высокой точностью и вычисления проводятся для чисел с предельным количеством значащих цифр, необходимых для компьютерного представления вещественных чисел двойной точности (8 байт). Для первого эксперимента эти отличия будут незначительными и связаны они с расчетами при $\varepsilon_x = 10^{-10}$. Для второго эксперимента, где используется $\varepsilon_x = 10^{-11}$, различия могут быть более значительными, но качественная картина вычислительного эксперимента не должна измениться.

Третий эксперимент связан с анализом разрешимости так называемой линейной задачи о допусках для интервальных систем линейных уравнений [17]. Рассматривается интервальная система линейных алгебраических уравнений вида $\boldsymbol{A}x = \boldsymbol{b}$ с интервальной $m \times n$ матрицей $\boldsymbol{A} = (\boldsymbol{a}_{ij})_{i,j=1}^{m,n}$ и интервальным m-вектором $\boldsymbol{b} = (\boldsymbol{b}_i)_{i=1}^{m}$. *Допусковым множеством решений* системы $\boldsymbol{A}x = \boldsymbol{b}$ называется множество, образованное всеми такими векторами $x \in \mathbb{R}^n$, что произведение Ax попадает в $\boldsymbol{b}$ для любого $A \in \boldsymbol{A}$. Линейной задачей о допусках называется задача нахождения (по возможности, большего) бруса, который содержится в допусковом множестве решений системы $\boldsymbol{A}x = \boldsymbol{b}$. Но в общем случае допусковое множество решений для интервальной системы линейных алгебраических уравнений может оказаться пустым.

Для заданной интервальной системы уравнений $\boldsymbol{A}x = \boldsymbol{b}$ введем *распознающий функционал* $\mathrm{Tol}(x)$ допускового множества решений, определив его выражением [18]

$$\mathrm{Tol}(x) = \mathrm{Tol}(x; \boldsymbol{A}, \boldsymbol{b}) = \min_{1 \le i \le m} \left\{ \mathrm{rad}\ \boldsymbol{b}_i - \left| \mathrm{mid}\ \boldsymbol{b}_i - \sum_{j=1}^{n} \boldsymbol{a}_{ij} x_j \right| \right\},$$

где rad — радиус интервала, mid — середина интервала. Принадлежность точки x допусковому множеству определяется неотрицательностью в ней функционала Tol, т. е. выполнением неравенства $\mathrm{Tol}(x; \boldsymbol{A}, \boldsymbol{b}) \ge 0$. Кроме того, распознающий функционал Tol является вогнутым относительно x и достигает по этой переменной конечного максимума на всем пространстве $\mathbb{R}^n$ (см. подробности в [17, 18]). Таким образом, с помощью распознающего функционала можно определить разрешимость линейной задачи о допусках — пустоту или непустоту допускового множества решений для рассматриваемой интервальной линейной системы $\boldsymbol{A}x = \boldsymbol{b}$, если исследовать знак безусловного максимума функционала Tol, т. е. величины $\max_{x \in \mathbb{R}^n} \mathrm{Tol}(x; \boldsymbol{A}, \boldsymbol{b})$. В случае, когда этот максимум неотрицателен, допусковое множество решений непусто, а иначе — пусто.

Третий эксперимент состоит в исследовании сходимости метода `ralgb4` для минимизации функции `shary` — выпуклой кусочно-линейной функции $f(x) = -\mathrm{Tol}(x)$ для интервальной линейной 7×7-системы с матрицей Ноймайера

$$\begin{pmatrix} 10.5 & [0,2] & \cdots & [0,2] \\ [0,2] & 10.5 & \cdots & [0,2] \\ \vdots & \vdots & \ddots & \vdots \\ [0,2] & [0,2] & \cdots & 10.5 \end{pmatrix} x = \begin{pmatrix} [-1,1] \\ [-1,1] \\ \vdots \\ [-1,1] \end{pmatrix}.$$

Ее минимальное значение $f^* = -1$ достигается в нулевой точке. В табл. 2 приведены результаты минимизации функции `shary` с помощью метода `ralgb4` при трех значениях коэффициента растяжения $\alpha \in \{2, 3, 4\}$ и пяти значениях коэффициента уменьшения шага $q_1 \in \{1.0, 0.95, 0.9, 0.85, 0.8\}$ (всего 15 вариантов расчета). По каждому варианту результаты программы `ralgb4` даны для шести значений $\varepsilon_x \in \{10^{-6}, 10^{-5}, \ldots, 10^{-1}\}$ и включают затраты на нахождение минимума $f(x)$ по количеству итераций `itn` и количеству вычислений значения функции и ее субградиента `nfg`, отклонение найденного рекордного значения f_r от оптимального $f^* = -1$. Остальные параметры программы `ralgb4` такие: $x_0 = (1, \ldots, 1)^T \in \mathbb{R}^7$, $h_0 = 1$, $n_h = 3$, $q_2 = 1.1$. Для вычисления значения кусочно-линейной вогнутой функции $\mathrm{Tol}(x)$ и ее суперградиента использовалась Octave-функция `calcfg`, реализованная С. П. Шарым.

Из табл. 2 видно, что для решения задачи `shary` метод `ralgb4` работает устойчиво, так как последовательное уменьшение ε_x в 10 раз не приводит к резким увеличениям количества итераций ни в одном из 15 вариантов расчета. Для нахождения минимума с точностью $\varepsilon_x = 10^{-6}$ наименьшие затраты для метода `ralgb4` составляют 69 `itn` и 112 `nfg` и реализуются при $\alpha = 2$, $q_1 = 0.8$, а наибольшие затраты — составляют 143 `itn` и 176 `nfg` и реализуются при $\alpha = 2$, $q_1 = 1.0$. Максимальное количество `nfg`, равное 214, реализуется при $\alpha = 4$, $q_1 = 0.8$. Следовательно, согласовывая настройку коэффициента растяжения α и коэффициента уменьшения шага q_1 при минимизации функции `shary`, можно добиться ускорения метода `ralgb4`, а значит, и метода `ralgb5`.

Т а б л и ц а 2. Затраты метода `ralgb4` в 7-мерной задаче `shary` с матрицей Ноймайера

	$\alpha = 2.0$, $q_1 = 1.0$			$\alpha = 3.0$, $q_1 = 1.0$			$\alpha = 4.0$, $q_1 = 1.0$		
ε_x	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$
1.0e-001	28	42	3.5e-001	20	32	6.7e-001	16	33	5.0e-001
1.0e-002	52	71	2.6e-002	35	54	6.2e-002	31	61	1.1e-001
1.0e-003	72	95	3.9e-003	48	74	8.4e-003	43	76	1.1e-002
1.0e-004	100	129	3.0e-004	69	116	5.0e-004	56	99	6.3e-004
1.0e-005	126	159	2.9e-005	87	143	4.3e-005	68	117	4.2e-005
1.0e-006	143	179	5.0e-006	102	168	4.1e-006	81	138	5.1e-006
	$\alpha = 2.0$, $q_1 = 0.95$			$\alpha = 3.0$, $q_1 = 0.95$			$\alpha = 4.0$, $q_1 = 0.95$		
ε_x	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$
1.0e-001	21	32	1.9e-001	20	38	5.6e-001	18	40	1.3e+000
1.0e-002	40	57	2.2e-002	33	61	5.7e-002	30	66	8.1e-002
1.0e-003	55	74	1.5e-003	47	81	4.2e-003	44	93	5.0e-003
1.0e-004	74	100	1.8e-004	61	104	3.7e-004	55	116	5.3e-004
1.0e-005	88	117	3.6e-005	72	117	5.2e-005	63	130	1.6e-004
1.0e-006	103	136	7.0e-006	84	135	9.0e-006	81	172	3.3e-006
	$\alpha = 2.0$, $q_1 = 0.9$			$\alpha = 3.0$, $q_1 = 0.9$			$\alpha = 4.0$, $q_1 = 0.9$		
ε_x	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$
1.0e-001	18	32	5.4e-001	17	34	1.1e+000	18	43	7.4e-001
1.0e-002	33	53	3.3e-002	31	58	8.0e-002	26	56	1.3e-001
1.0e-003	45	67	4.7e-003	42	77	7.2e-003	37	78	2.1e-002
1.0e-004	57	81	2.4e-004	56	100	6.0e-004	52	119	4.6e-004
1.0e-005	71	96	3.3e-005	65	115	1.1e-004	61	136	1.7e-004
1.0e-006	81	107	3.7e-006	83	152	4.7e-006	75	165	8.9e-006
	$\alpha = 2.0$, $q_1 = 0.85$			$\alpha = 3.0$, $q_1 = 0.85$			$\alpha = 4.0$, $q_1 = 0.85$		
ε_x	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$
1.0e-001	17	30	1.8e-001	13	26	4.6e-001	17	39	8.6e-001
1.0e-002	29	45	2.3e-002	25	48	7.5e-002	24	55	1.7e-001
1.0e-003	39	58	3.3e-003	39	73	1.9e-003	35	84	7.7e-003
1.0e-004	50	74	2.8e-004	47	85	5.5e-004	46	106	1.3e-003
1.0e-005	64	96	3.3e-005	55	95	7.6e-005	58	130	1.2e-004
1.0e-006	75	113	4.9e-006	65	110	6.6e-006	72	172	1.6e-005
	$\alpha = 2.0$, $q_1 = 0.8$			$\alpha = 3.0$, $q_1 = 0.8$			$\alpha = 4.0$, $q_1 = 0.8$		
ε_x	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$	itn	nfg	$f_r - f^*$
1.0e-001	15	28	7.7e-001	15	31	4.8e-001	15	40	6.8e-001
1.0e-002	25	44	1.2e-001	29	63	6.9e-002	24	58	1.2e-001
1.0e-003	39	66	7.0e-003	39	86	9.0e-003	34	85	1.1e-002
1.0e-004	49	81	1.1e-003	48	99	7.2e-004	44	115	3.2e-003
1.0e-005	57	95	7.4e-005	56	115	5.0e-005	58	173	2.4e-004
1.0e-006	69	112	4.3e-006	67	136	1.6e-005	74	214	7.2e-006

Для 7-мерной задачи `shary` с матрицей Ноймайера предпочтительными вариантами будут следующие: $\alpha = 2$, $q_1 = 0.8$ (69 `itn` и 112 `nfg`) и $\alpha = 4$, $q_1 = 1.0$ (81 `itn` и 138 `nfg`).

Подобный эффект будет иметь место и для интервальной линейной системы, когда матрица Ноймайера имеет другие размеры, отличные от 7. Так, например, 4-мерной функции `shary` отвечает выпуклая кусочно-линейная функция $f(x) = -\mathrm{Tol}(x)$ для интервальной линейной 4×4-системы

$$\begin{pmatrix} 5.5 & [0,2] & [0,2] & [0,2] \\ [0,2] & 5.5 & [0,2] & [0,2] \\ [0,2] & [0,2] & 5.5 & [0,2] \\ [0,2] & [0,2] & [0,2] & 5.5 \end{pmatrix} x = \begin{pmatrix} [-1,1] \\ [-1,1] \\ [-1,1] \\ [-1,1] \end{pmatrix},$$

минимальное значение которой $f^* = -1$. Для нее обсуждаемые выше варианты расчета по нахождению минимума с точностью $\varepsilon_x = 10^{-6}$ показывают такие результаты: $\alpha = 2$, $q_1 = 1.0$ (79 `itn` и 112 `nfg`), $\alpha = 4$, $q_1 = 1.0$ (43 `itn` и 71 `nfg`), $\alpha = 2$, $q_1 = 0.8$ (49 `itn` и 72 `nfg`). Понятно, что предпочтительными для 4-мерной функции `shary` будут два последних варианта, при этом лучше выбрать вариант, для которого $\alpha = 4$, $q_1 = 1.0$.

Из табл. 2 легко видеть, что для 14 вариантов из 15 (кроме $\alpha = 3$, $q_1 = 0.9$) величина $f_r - f^* < 1$ уже при $\varepsilon_x = 10^{-1}$. Это доказывает, что линейная задача о допусках для интервальной линейной 7×7-системы разрешима. Действительно, из того, что $f(x_r) - f^* < 1$ и $f^* = -1$, для найденного x_r получаем неравенство $f(x_r) < 0$, которое равносильно неравенству $\mathrm{Tol}(x_r) > 0$, доказывающему принадлежность точки x_r допусковому множеству. Если $\alpha = 2$ и $q_1 = 0.8$, то согласно табл. 2 затраты на доказательство составляют 15 `itn` и 28 `nfg`. На самом деле доказательство получено уже на седьмой итерации, о чем свидетельствует фрагмент

```
itn    0 f  2.15000000e+001 fr  2.1500000000000e+001 ls  0 ncalls    1
itn    1 f  1.70458320e+001 fr  1.2422877627166e+001 ls  3 ncalls    4
itn    2 f  6.39881977e+000 fr  4.6437447981195e-001 ls  4 ncalls    8
itn    3 f  4.64374480e-001 fr  4.6437447981195e-001 ls  2 ncalls   10
itn    4 f  4.77081604e+000 fr  4.6437447981195e-001 ls  1 ncalls   11
itn    5 f  2.20674999e-002 fr  2.2067499873478e-002 ls  2 ncalls   13
itn    6 f  3.73740074e+000 fr  2.2067499873478e-002 ls  1 ncalls   14
itn    7 f -2.33825570e-001 fr -2.3382556976340e-001 ls  2 ncalls   16
```

из протокола работы программы `ralgb4`, где на седьмой итерации получена точка x_r, для которой $f(x_r) < 0$. При этом доказательство получено за 7 `itn` и 16 `nfg`.

Следовательно, для анализа разрешимости интервальной системы $\boldsymbol{A}x = \boldsymbol{b}$ целесообразно разрабатывать двухэтапные алгоритмы: на первом этапе выясняем вопрос о разрешимости линейной задачи о допусках, а на втором этапе находим параметры для бруса, который находится в допусковом множестве решений системы $\boldsymbol{A}x = \boldsymbol{b}$. Для первого этапа можно использовать субградиентный метод `amsg2p`, который использует релаксационный шаг (известен как шаг Поляка или шаг Агмона — Моцкина) и априорное знание минимального значения функции [13, с. 294–297]. В методе применяется антиовражная техника подобно тому, как это сделано в r-алгоритмах Шора. Преобразование пространства реализуется с помощью однорангового линейного оператора, который позволяет обеспечить уменьшение объема локализующего точку минимума эллипсоида в каждом последующем преобразованном пространстве переменных. При этом в методе `amsg2p` для неовражных функций будет выполняться меньше преобразований пространства, чем для овражных или существенно овражных функций.

Заключение

Субградиентные методы `ralgb5` и `ralgb4` можно использовать при решении негладких задач оптимизации, возникающих в различных областях математического моделирования. Так как гладкая функция с очень быстро изменяющимся градиентом близка

по своим свойствам к негладкой функции, они обладают ускоренной сходимостью при оптимизации овражных гладких функций. Матрично-векторные вычисления для обоих методов легко поддаются параллельной обработке, что может быть полезным при их реализации на параллельных ЭВМ.

Octave-функции `ralgb5` и `ralgb4` можно использовать как оптимизационные ядра при реализации на языке Octave алгоритмов решения задач нелинейного программирования. На их основе легко разрабатывать оптимизационные ядра на языке MATLAB для решения вычислительных задач, которые сводятся к минимизации негладких выпуклых функций или гладких выпуклых функций с овражной структурой поверхностей уровня. Octave-функции `ralgb5` и `ralgb4` легко переписать на языки Фортран и Си, используя библиотеку базовых подпрограмм линейной алгебры BLAS (Basic Linear Algebra Subprograms) или библиотеку математических прикладных программ Intel Math Kernel Library (Intel MKL), которые оптимизированы под современные вычислительные машины. Это может позволить значительно ускорить решение больших задач, имеющих тысячу и более переменных. Так, например, одну и ту же задачу с 1000 переменными метод `ralgb5` для GNU Octave версии 3.6.4 с использованием библиотеки BLAS решает в два раза быстрее, чем исполняемый оптимизируемый код компилятора Visual Studio C++ 2008 Express Edition без использования библиотеки BLAS. Эксперимент проводился на компьютере с процессором Pentium 3GHz в операционной системе Windows 7 (32-битная версия).

С помощью программ `ralgb5` и `ralgb4` 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.

Благодарности. Работа выполнена при поддержке НАНУ (проекты № 0114U001055 и № 0116U004558) и Volkswagen Foundation (грант № 90 306).

Автор признателен С.П. Шарому за активное использование r-алгоритмов в задачах интервального анализа и Е.А. Нурминскому за то, что восемь лет тому назад убедил автора перейти на программирование под GNU Octave.

Список литературы / References

[1] **Шор Н.З.** Методы минимизации недифференцируемых функций и их применения. Киев: Наук. думка, 1979. 200 с.
Shor, N.Z. Minimization Methods for Non-Differentiable Functions. Kiev: Naukova Dumka, 1979, 200 p. (In Russ.)

[2] **Shor, N.Z.** Nondifferentiable optimization and polynomial problems. Boston; Dordrecht; London: Kluwer Acad. Publ. 1998. 412 p.

[3] **Шор Н.З.** Методы минимизации недифференцируемых функций и их приложения: Автореф. дис. ... докт. физ-мат. наук. Киев, 1970. 44 с.
Shor, N.Z. Minimization methods for non-differentiable functions and their applications: Avtoref. Dis. ... Dokt. fiz-mat. Nauk. Kiev, 1970. 44 p. (In Russ.)

[4] **Шор Н.З., Журбенко Н.Г.** Метод минимизации, использующий операцию растяжения пространства в направлении разности двух последовательных градиентов // Кибернетика. 1971. № 3. С. 51–59.
Shor, N.Z., Zhurbenko, N.G. A minimization method using the operation of extension of the space in the direction of the difference of two successive gradients // Cybernetics and Systems Analysis. 1971. Vol. 7, No. 3. P. 450–459.

[5] GNU Octave. Available at: http://www.octave.org

[6] **Стецюк П.И.** К вопросу сходимости r-алгоритмов // Кибернетика и систем. анализ. 1995. № 6. С. 173–177.
Stetsyuk, P.I. Convergence of r-algorithms // Cybernetics and Systems Analysis. 1995. Vol. 31, No. 6. P. 935–937.

[7] **Стецюк П.И., Ивличев А.В., Ищенко А.А.** О сходимости $r_\mu(\alpha)$-алгоритма // Компьютерная математика. 2015. № 1. С. 142–152.
Stetsyuk, P.I., Ivlichev, A.V., Ishchenko, A.A. On the convergence of $r_\mu(\alpha)$-algorithm // Computer Mathematics. 2015. No. 1. P. 142–152. (In Russ.)

[8] **Шор Н.З., Стеценко С.И.** Квадратичные экстремальные задачи и недифференцируемая оптимизация. Киев: Наук. думка, 1989. 208 с.
Shor, N.Z., Stetsenko, S.I. Quadratic extremal problems and non-differentiable optimization. Kiev: Naukova Dumka, 1989. 208 p. (In Russ.)

[9] **Журбенко Н.Г., Марчук Т.В.** Алгоритм минимизации негладких функционалов ($r(\alpha)$-алгоритм). АН УССР, РФАП, 1976. № 22.
Zhurbenko, N.G., Marchuk, T.V. Algorithm for minimization of nonsmooth functionals ($r(\alpha)$-algorithm). AN USSR, RFAP, 1976. No. 22. (In Russ.)

[10] **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.
Shor, N.Z., Stetsyuk, P.I. Modified r-algorithm to find the global minimum of polynomial functions // Cybernetics and Systems Analysis. 1997. Vol. 33, No. 4. P. 482–497.

[11] **Kappel, F., Kuntsevich, A.V.** An implementation of Shor's r-algorithm // Comput. Optim. and Appl. 2000. Vol. 15, No. 2. P. 193–205.

[12] **Стецюк П.И., Лиховид А.П., Чумаков Б.М., Видил А.Ю., Пилиповский А.В.** Математические и программные средства моделирования и оптимизации динамической загрузки мощностей энергосистемы: Отчет о НИР. № гос. рег. 0107U004963. Киев: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2009. 136 с. Адрес доступа: `http://www.incyb.kiev.ua/file/d120-energy/Ot2009-2mb.pdf`
Stetsyuk, P.I., Likhovid, A.P., Chumakov, B.M., Vidil, A.Yu., Pilipovskiy, A.V. Mathematical and software tools for modeling and optimization of dynamic loading of power system capacity: Otchet o NIR. № gos. reg. 0107U004963. Kiev: In-t kibernetiki im. V.M. Glushkova NAN Ukrainy, 2009. 136 p. (In Russ.) Available at: http://www.incyb.kiev.ua/file/d120-energy/Ot2009-2mb.pdf

[13] **Стецюк П.И.** Методы эллипсоидов и r-алгоритмы. Кишинэу: Эврика, 2014. 488 с.
Stetsyuk, P.I. Ellipsoids methods and r-algorithms. Kishineu: Evrika, 2014. 488 p. (In Russ.)

[14] `tolsolvty` — программа для исследования разрешимости интервальной линейной задачи о допусках. Адрес доступа: `http://www.nsc.ru/interval/Programing/SciCodes`
`tolsolvty` — a program for investigation of solvability of the interval linear tolerance problem. Available at: http://www.nsc.ru/interval/Programing/SciCodes

[15] **Lemareshal, C., Mifflin, R.** Nonsmooth optimization. Oxford: Pergamon Press, 1978. 186 p.

[16] **Ржевский С.В.** Монотонные методы выпуклого программирования. Киев: Наук. думка, 1993. 324 с.
Rzhevskiy, S.V. Monotonic methods of convex programmimg. Kiev: Naukova Dumka, 1993. 324 p. (In Russ.)

[17] **Шарый С.П.** Решение интервальной линейной задачи о допусках // Автоматика и телемеханика. 2004. № 7. С. 147–162.
Shary, S.P. Solving interval linear tolerance problem // Avtomatika i Telemekhanika. 2004. No. 7. P. 147–162. (In Russ.)

[18] **Shary, S.P.** Solving the linear interval tolerance problem // Mathematics and Computers in Simulation. 1995. Vol. 39. P. 53–85.

Поступила в редакцию 28 ноября 2017 г.

Subgradient methods ralgb5 and ralgb4 for minimization of ravine-like convex functions

Stetsyuk, Petr I.

V.M. Glushkov Institute of Cybernetics of Ukraine NAS, Kiev, Ukraine

Corresponding author: Stetsyuk, Petr I. e-mail: `stetsyukp@gmail.com`

We consider properties of the three computational forms of the r-algorithm proposed by N.Z. Shor for optimization of non-smooth functions that differ in the complexity of a single iteration. Discussed is a variant of the r-algorithm with adaptive stepsize control along the direction of the normalized antisubgradient in the transformed space of variables. The Octave functions `ralgb5` and `ralgb4` are described, which implement two computationally stable forms of the r-algorithms with adaptive stepsize control and a constant space dilation factor. The results of computational experiments for an essentially ravine-like piecewise quadratic function and a piecewise linear function related to solvability of interval linear tolerance problem are presented.

Keywords: subgradient method, space dilation, r-algorithm, adaptive stepsize control, GNU Octave, Octave function, piecewise-quadratic function `maxquad`, interval linear tolerance problem.

Acknowledgements. This work was supported by NASU (projects No. 0114U001055, No. 0116U004558) and Volkswagen Foundation (grant No. 90 306).

The author is grateful to S.P. Shary for the active use of r-algorithms in interval analysis problems and E.A. Nurminsky who convinced the author to switch to programming under the GNU Octave eight years ago.

Received 28 November 2016