Субградиентные методы с преобразованием пространства для минимизации овражных выпуклых функций

П.И. Стецюк
Институт кибернетики им. В.М.Глушкова НАН Украины
e-mail: stetsyukp@gmail.com

Приведены Octave-функция ralgb5 (реализует r-алгоритм с адаптивной регулировкой шага и постоянным коэффициентом растяжения пространства) и алгоритм-функция amsg2p (реализует релаксационный метод с преобразованием пространства, которое использует два последовательных субградиента и агрегатный вектор, являющийся выпуклой комбинацией вычисленных ранее субградиентов). Даны результаты экспериментов для существенно овражной кусочно-квадратичной функции и кусочно-линейной функции, которая связана с разрешимостью интервальной линейной задачи о допусках.

Функции с разрывным градиентом часто встречаются в различных задачах интервального анализа как следствие кусочной гладкости интервальных арифметических операций. Поэтому владение численными методами минимизации негладких выпуклых функций дает разработчику действенный инструмент при построении эффективных алгоритмов для решения задач математического программирования в интервальном анализе. Подробное изложение основных источников, порождающих задачи негладкой оптимизации, и приложение к ним методов минимизации недифференцируемых функций можно найти в [1]. Очень информативны на этот счет сборники научных трудов [2, 3], в которые включены основные работы Н.З. Шора по методам негладкой оптимизации и их применению в задачах большой размерности, квадратичного, булевого, матричного и стохастического программирования. Сборники доступны в электронном варианте по адресу http://elis.dvo.ru/?q=node/114 (режим доступа — свободный).

Негладкие выпуклые функции, как правило, характеризуются овражной или существенно овражной структурой поверхностей уровня. Качество методов минимизации негладких 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 функций. Ниже обсудим алгоритмы из двух семейств субградиентных методов оптимизации с преобразованием пространства, имеющих ускоренную сходимость при минимизации выпуклых функций с овражной структурой поверхностей уровня.

Первый метод — $r(\alpha)$-алгоритм (реализован функцией ralgb5 на языке Octave [4]) относится к семейству субградиентных методов минимизации негладких функций, которое известно как r-алгоритмы Н.З. Шора. r-Алгоритмы базируются на процедуре наискорейшего спуска в преобразованном пространстве переменных и обеспечивают монотонность (или почти монотонность) по значениям минимизируемой функции. r-Алгоритмы используют операцию растяжения пространства в направлении разности

двух последовательных субградиентов, которая улучшает свойства овражной функции в преобразованном пространстве переменных.

Второй метод (реализован алгоритм-функцией amsg2p) относится к семейству методов минимизации негладких функций, которые используют релаксационный шаг (известен как шаг Поляка или шаг Агмона-Моцкина), и использует априорное знание минимального значения функции. Здесь применяется антиовражная техника, подобно тому, как это сделано в r-алгоритмах Шора. Но преобразование пространства реализуется с помощью линейного оператора [5], который позволяет обеспечить уменьшение расстояния до точки минимума в очередном преобразованном пространстве переменных.

1. Octave-функция ralgb5

Программа находит точку минимума x_r^* выпуклой функции $f(x)$ от n переменных и делает это с помощью $r(\alpha)$-алгоритма — вариант r-алгоритмов с постоянным на каждой итерации коэффициентом растяжения пространства α ($\alpha>1$) и адаптивной регулировкой шага в направлении нормированного антисубградиента. Программа использует Octave-функцию function [f,g] = calcfg(x), которая вычисляет значение функции $f = f(x)$ и её субградиента $g = \partial f(x)$ в точке x. Программа использует следующие параметры.

```
%  Входные параметры:
%    calcfg -- имя функции вида calcfg(x) для вычисления f и g
%    x  -- начальная точка x(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(n)
%    fr -- значение функции в точке минимума
%    itn -- число затраченных итераций
%    ncalls -- число вызовов функции calcfg
%    istop -- код останова (2 = epsg, 3 = epsx, 4 = maxitn, 5 = error)
```

Адаптивная регулировка шага в $r(\alpha)$-алгоритме выполняет одномерный спуск в направлении нормированного антисубградиента в преобразованном пространстве переменных и реализуется 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 ($q_2\geq 1$); натуральное число n_h задает число шагов одномерного спуска ($n_h > 1$), через каждые из которых шаг будет увеличиваться в q_2 раз. Подробные рекомендации по выбору коэффициента растяжения пространства и параметров адаптивной регулировки шага даны в [6], с. 45–47. Их суть 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 быть большим.

Параметры ε_x и ε_g определяют условия завершения $r(\alpha)$-алгоритма: метод останавливается в точке x_{k+1}, если выполнено $\|x_{k+1} - x_k\| \leq \varepsilon_x$ (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). Аварийное завершение программы связано либо с тем, что функция $f(x)$ неограничена снизу, либо h_0 слишком мал и его требуется увеличить.

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

При минимизации негладких функций рекомендуется следующий выбор параметров: $\alpha = 2 \div 3$, $h_0 = 1.0$, $q_1 = 1.0$, $q_2 = 1.1 \div 1.2$, $n_h = 2 \div 3$. 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 обеспечивает нахождение

x_r^* со значением функции, достаточно близким к оптимальному. При этом обычно

$$\frac{f(x_r^*) - f(x^*)}{|f(x^*)| + 1} \sim 10^{-6} \div 10^{-5} \text{ — для негладких}$$

и

$$\frac{f(x_r^*) - f(x^*)}{|f(x^*)| + 1} \sim 10^{-12} \div 10^{-10} \text{ — для гладких функций,}$$

что подтверждается результатами многочисленных тестовых и реальных расчетов.

2. Алгоритм-функция amsg2p

Алгоритм amsg2p находит точку минимума выпуклой функции $f(x)$ при известном её минимальном значении f^*. В его основу положен второй из субградиентных методов с преобразованием пространства и регулировкой шага Агмона-Моцкина-Шонберга (AMS-шаг) в преобразованном пространстве переменных [7]. Алгоритм использует величину максимального сдвига по выпуклости функции $f(x)$ (задается параметром γ), для которой субградиент $\partial f(x)$ удовлетворяет следующему условию:

$$\langle x - x^*, \partial f(x) \rangle \geq \gamma(f(x) - f^*), \quad \text{где } \gamma \geq 1, \tag{1}$$

для любого $x \in \mathbb{R}^n$ и произвольного x^* из множества X^* точек минимума функции $f(x)$. Неравенство (1) позволяет реализовать более сильные AMS-шаги для специальных классов функций. Так, например, для квадратичной гладкой функции рекомендуется использовать $\gamma = 2$.

Алгоритм-функция amsg2p: $(x_\varepsilon^*, k_\varepsilon^*) = \text{amsg2p}(x_0, \varepsilon, f^*, \gamma)$

На итерации $k = 0$ имеем начальное приближение $x_0 \in \mathbb{R}^n$ и достаточно малое $\varepsilon > 0$. Вычислим $f(x_0)$ и $\partial f(x_0)$. Если $f(x_0) - f^* \leq \varepsilon$, то $x_\varepsilon^* = x_0$, $k_\varepsilon^* = 0$ и окончание работы алгоритма. Иначе положим

$$h_0 = \frac{\gamma(f(x_0) - f^*)}{\|\partial f(x_0)\|}, \qquad \xi_0 = \frac{\partial f(x_0)}{\|\partial f(x_0)\|} \in \mathbb{R}^n, \qquad p_0 = 0 \in \mathbb{R}^n,$$

$B_0 = I_n$ — единичная матрица размера $n \times n$. Перейдем к следующей итерации.

Пусть на k-й итерации получены $x_k \in \mathbb{R}^n$, h_k, $\xi_k \in \mathbb{R}^n$, $p_k \in \mathbb{R}^n$, B_k — матрица $n \times n$. Для $(k+1)$-й итерации выполним пп. 1–5.

1. Вычислим очередное приближение

$$x_{k+1} = x_k - h_k B_k \xi_k.$$

2. Вычислим $f(x_{k+1})$ и $\partial f(x_{k+1})$. Если $f(x_{k+1}) - f^* \leq \varepsilon$, то $x_\varepsilon^* = x_{k+1}$, $k_\varepsilon^* = k+1$ и окончание алгоритма. Иначе положим

$$\xi_{k+1} = \frac{B_k^T \partial f(x_{k+1})}{\|B_k^T \partial f(x_{k+1})\|}, \quad h_{k+1} = \frac{\gamma(f(x_{k+1}) - f^*)}{\|B_k^T \partial f(x_{k+1})\|}.$$

3. Вычислим $\lambda_1 = -p_k^T \xi_{k+1}$ и $\lambda_2 = -\xi_k^T \xi_{k+1}$. Положим

$$p_{k+1} = \begin{cases} \dfrac{\lambda_1}{\sqrt{\lambda_1^2 + \lambda_2^2}}\, p_k + \dfrac{\lambda_2}{\sqrt{\lambda_1^2 + \lambda_2^2}}\, \xi_k, & \text{если } \lambda_1 > 0 \text{ и } \lambda_2 > 0, \\ p_k, & \text{если } \lambda_1 > 0 \text{ и } \lambda_2 \le 0, \\ \xi_k, & \text{если } \lambda_1 \le 0 \text{ и } \lambda_2 > 0, \\ 0, & \text{если } \lambda_1 \le 0 \text{ и } \lambda_2 \le 0. \end{cases}$$

4. Вычислим $\mu_k = p_{k+1}^T \xi_{k+1}$. Если $-0.98 \le \mu_k \le 0$, то вычислим

$$B_{k+1} = B_k + (B_k \eta)\, \xi_{k+1}^T, \quad \text{где} \quad \eta = \left(\frac{1}{\sqrt{1-\mu_k^2}} - 1 \right) \xi_{k+1} - \frac{\mu_k}{\sqrt{1-\mu_k^2}} p_{k+1}$$

и пересчитаем

$$h_{k+1} = \frac{h_{k+1}}{\sqrt{1-\mu_k^2}}, \qquad p_{k+1} = \frac{1}{\sqrt{1-\mu_k^2}} (p_{k+1} - \mu_k \xi_{k+1}).$$

Иначе положим $B_{k+1} = B_k$ и $p_{k+1} = 0$.

5. Перейдем к следующей итерации с x_{k+1}, h_{k+1}, ξ_{k+1}, p_{k+1}, B_{k+1}.

3. Вычислительный эксперимент

Использование Octave-функции ralgb5 проиллюстрируем для задачи maxquad [8], которая связана с минимизацией существенно овражной выпуклой кусочно-квадратичной функции $\varphi(x)$ от 10 переменных. Здесь $\varphi(x) = \max\limits_{1 \le k \le 5} f_k(x)$, 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 $A_{kii} = i\,|\sin k|\,/10 + \sum\limits_{j \ne i} |A_{kij}|$, а компоненты векторов b_k определяются $b_{ki} = e^{i/k} \sin(ik)$. В качестве начального приближения maxquad использует точку $x^0 = (1, \ldots, 1)^T \in \mathbb{R}^{10}$. Подготовку данных для задачи maxquad и вызов ralgb5 реализует такой 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
alpha = 2, h0 = 1.0, nh = 3, q1 = 1.0, q2 = 1.1
epsx = 1.e-6, epsg = 1.e-6, maxitn = 1000, x0 = ones(n,1)
[xr,fr,itn,ncalls,istop]=ralgb5(@maxquad,x0,alpha,h0,q1,q2,nh,epsg,epsx,maxitn);
xr,fr,itn,ncalls,istop
```

в котором вычисление $\varphi(x)$ и $\partial\varphi(x)$ выполняется с помощью Octave-функции maxquad

```
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
```

Затраты $r(\alpha)$-алгоритма и метода amsg2p для нахождения в задаче maxquad единственного решения с достаточно высокой точностью позволяет оценить приведенный ниже фрагмент численных расчетов с обеими программами.

```
Maxquad: f(x0)    5.3370664293114e+003 fmin = -8.4140833459641e-001 gamma=1
..epsx..  .....fr(itn)........  itn(ncalls)  ..epsf..  .....f(itn).........  .itn.
1.0e-001  -7.3721660556183e-001     35(40)   1.0e-001  -7.7355266120112e-001    17
1.0e-002  -8.3982079259954e-001     68(74)   1.0e-003  -8.4084776169123e-001    29
1.0e-003  -8.4138254819439e-001   107(117)   1.0e-004  -8.4132394277880e-001    35
1.0e-004  -8.4139971517765e-001   120(131)   1.0e-005  -8.4140078034524e-001    41
1.0e-005  -8.4140785230390e-001   148(164)   1.0e-006  -8.4140807664455e-001    49
1.0e-006  -8.4140830366048e-001   175(195)   1.0e-011  -8.4140833458913e-001    94
1.0e-007  -8.4140833400334e-001   211(236)   1.0e-012  -8.4140833459555e-001   101
1.0e-008  -8.4140833455704e-001   240(267)   1.0e-013  -8.4140833459633e-001   110
1.0e-009  -8.4140833459582e-001   278(309)   1.0e-014  -8.4140833459640e-001   116
1.0e-010  -8.4140833459641e-001   330(369)   1.0e-015  -8.4140833459641e-001   122
```

Как видим затраты по числу итераций на нахождение точки минимума при известном $f^* = fmin$ в несколько раз меньше, чем затраты программы ralgb5. Это может помочь при анализе разрешимости интервальных систем линейных уравнений [9]. Так, например, для интервальной линейной 7×7-системы с матрицей Ноймайера

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

на нахождение максимума распознающего функционала допускового множества решений [10] программа ralgb5 затратила 246 итераций, а алгоритм amsg2p всего за 30 итераций находит точку, где оптимальное значение (равно 1) реализуется с точностью $\varepsilon = 10^{-10}$. Для вычисления суперградиента кусочно-линейной вогнутой функции использовалась реализованная С.П. Шарым функция calcfg.

Приведенные алгоритмы 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 наши алгоритмы обладают ускоренной сходимостью при оптимизации овражных гладких функций. Матрично-векторные вычисления для обоих семейств алгоритмов легко поддаются параллельной обработке, что может быть полезным при их реализации на параллельных ЭВМ.

Список литературы

[1] Шор Н.З., Журбенко Н.Г., Лиховид А.П., Стецюк П.И. Развитие алгоритмов недифференцируемой оптимизации и их приложения // Кибернетика и Системный Фнализ. 2003. № 4. С. 80–94.

[2] Шор Н.З. Методы недифференцируемой оптимизации и сложные экстремальные задачи: Сб. избр. тр. Кишинэу: Эврика, 2008. 270 с.

[3] Шор Н.З. Методы минимизации негладких функций и матричные задачи оптимизации: Сб. избр. тр. Кишинэу: Эврика, 2009. 240 с.

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

[5] Стецюк П.И. Ортогонализующие линейные операторы в выпуклом программировании // Кибернетика и Системный Анализ. 1997. № 3. С. 97–119.

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

[7] Стецюк П.И. Субградиентные методы переменной метрики, использующие шаг Агмона-Моцкина и одноранговый эллипсоидальный оператор // Труды АТИК - 2007–2008. Кишинэу: Эврика, 2009. Том. I (XII). С. 16–25.

[8] Lemarechal C. Numerical experiments in nonsmooth optimization // Progress in nondifferentiable optimization / Ed. E.A. Nurminski. CP-82-58. International Institute for Applied System Analysis: Laxenburg, Austria, 1982. P. 61–84.

[9] Шарый С.П. Интервальный анализ или методы Монте-Карло? // Вычислительные Технологии. 2007. Том 12, № 1. С. 103–115.

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