运筹学与控制论

DOI:10.11721/cqnuj20140402

一种极小化两个凸函数之和的混合近似邻近点算法*

陈雍梅，白富生
(重庆师范大学 数学学院，重庆 401331)

摘要:本文提出一种混合近似邻近点算法以求解极小化两个凸函数之和的无约束优化问题。通过将邻近点算法中的优化问题转化为一系列极小化近似函数的子问题来求解,以得到此优化问题的最优解。在子问题中用线性模型来取代原问题目标函数中非线性程度较低的函数,而在下一个子问题中,用二次模型来取代非线性程度较高的函数,进行交替运算。在临近点算法的框架下,求出原问题的解。最后给出 3 个算例以说明本文所给出的算法是有效的。

关键词:凸规划;近似邻近点算法;线性模型;二次模型

中图分类号:O221.2　　**文献标志码**:A　　**文章编号**:1672-6693(2014)04-0006-06

1 问题介绍

本文考虑如下形式的优化问题:

$$\min F(x) := h(x)+f(x) \tag{1}$$

其中 $h,f:\mathbf{R}^n\to\mathbf{R}$ 是二次连续可微凸函数。

B. Martinet 和 R. T. Rockafellar 最早提出了邻近点算法(Proximal point algorithm)[1-2],记为 PPA。此算法考虑的问题如下:

$$\min_{x\in\mathbf{R}^n} f(x)$$

其中 $f:\mathbf{R}^n\to(-\infty,+\infty]$为闭真凸函数。记 $f^*=\inf\limits_{x\in\mathbf{R}^n} f(x)$,$X^*=\arg\min\limits_{x\in\mathbf{R}^n} f(x)$。设 X^* 非空。

任意选取一个初始点 $x^1\in\mathbf{R}^n$,由 $x^{k+1}=\arg\min\limits_x f(x)+\frac{1}{2}\beta_k\|x-x^k\|^2$,$k=1,2,3,\cdots$,产生一个序列$\{x_k\}$。这里$\|\cdot\|$为欧几里得范数,$\{\beta_k\}$是一个正数序列。此方法称为邻近点算法。文献[3]给出了邻近点算法的收敛性结论:设$\{x^k\}$为邻近点算法产生的序列,若 $\sum\limits_{k=1}^{\infty}\beta_k=\infty$,则 $f(x^k)$ 单调递减趋向于 f^* ,且$\{x_k\}$收敛于 X^* 中的某点。文献[4-5]给出了关于近似点方法的一些新的结果。

为了解决问题(1),文献[6]中的方法是通过邻近点算法将问题(1)转化成如下问题:

$$\min_x F(x)+\frac{1}{2}\beta_k\|x-x^k\|^2 \tag{2}$$

为了分别利用 $h(x)$和 $f(x)$的结构性质,文献[6]采用如下形式来求解问题(2):

$$\min_x \tilde{f}_k(x)+h(x)+\frac{1}{2}\beta_k\|x-x^k\|^2$$

和　$\min\limits_x f(x)+\tilde{h}_k(x)+\frac{1}{2}\beta_{k+1}\|x-x^{k+1}\|^2$

其中,$\tilde{f}_k$ 和 $\tilde{h}_k$ 分别是 f 和 h 的线性模型,称此方法为交替线性化方法(Alternating linearization method),记为

* **收稿日期**:2013-05-20　　**修回日期**:2013-10-11　　**网络出版时间**:2014-7-3 23:03
资助项目:重庆市自然科学基金资助项目(No. cstc2011jjA00010)
作者简介:陈雍梅,女,研究方向为最优化理论与算法,E-mail:332459164@qq.ocm;**通讯作者**:白富生,E-mail:fsbai@cqnu.edu.cn
网络出版地址:http://www.cnki.net/kcms/detail/50.1165.N.20140703.2303.002.html

ALM。文献[6]给出了 ALM 详细的收敛性证明。此外，文献[7]给出的并行 Uzawa 方法及文献[8]给出的快速交替线性化方法也可用来求解问题(1)。

ALM 有广泛应用，下面举例说明它在增广拉格朗日函数方法中的一种重要应用[6]。考虑一个带有耦合约束的可分问题[P]：

$$\min \sum_{j=1}^{N} \varphi_j(x_j)$$

$$\text{s. t.} \quad \sum_{j=1}^{N} \mathbf{A}_j x_j = b$$

其中 $\varphi_j: \mathbf{R}^{n_j} \to (-\infty, +\infty]$是闭真凸函数，$\mathbf{A}_j$ 是 $m \times n_j$ 的矩阵，$j=1,\cdots,N$。

应用乘子法[9-12]可以将问题[P]转化为极小化增广拉格朗日函数问题[P′]：

$$\min_x \sum_{j=1}^{N} (\varphi_j(x_j) - <\lambda, A_j x_j>) + <\lambda, b> + \frac{1}{2}\rho \| Ax - b \|^2$$

其中 $\lambda \in \mathbf{R}^m$ 是拉格朗日乘子向量，$\rho>0$ 是罚参数，$x=(x_1^{\mathrm{T}},\cdots,x_N^{\mathrm{T}})^{\mathrm{T}}$，$\mathbf{A}=(A_1,\cdots,A_N)$。

特别地，令 $f(x)=\frac{1}{2}\rho \| Ax-b \|^2$，$h(x)=\sum_{j=1}^{N}(\varphi_j(x_j) - <\lambda, A_j x_j>) + <\lambda, b>$，则问题[$P'$]具有形式(1)。注意到对$\frac{1}{2}\rho \| Ax-b \|^2$线性化后，相应的子问题可化为 N 个独立的子问题进行求解，由此可以利用原问题的可分结构。

考虑到如果函数 $f(x)$或者 $h(x)$的非线性程度较高，此时用 ALM 计算，计算结果误差可能会比较大。因此，本文用二阶泰勒展式去逼近非线性程度较高的函数，而用线性函数去逼近非线性程度较低的函数。不妨设 $h(x)$的非线性程度较高，则为了求解问题(2)，作者通过求解如下形式的子问题来进行：

$$\min_x \tilde{f}_k(x) + h(x) + \frac{1}{2}\| x - x^k \|_D^2$$

$$\text{和} \min_x f(x) + \tilde{h}_k(x) + \frac{1}{2}\| x - x^{k+1} \|_D^2$$

其中$\tilde{f}_k$ 和$\tilde{h}_k$ 分别是 f 和 h 的线性模型和二次模型，$\frac{1}{2}\| x-x^k \|_D^2 = \frac{1}{2}(x-x^k)^{\mathrm{T}} D (x-x^k)$，$\mathbf{D}$ 为对角矩阵且对角线上的值为正。通过求解一系列问题(2)而得到问题(1)的解。称此方法为混合近似临近点算法(Hybrid approximate proximal point algorithm)，记为 HAPPA。

本文结构安排如下：第 2 部分介绍混合近似临近点算法和交替线性化算法；第 3 部分给出应用混合近似临近点算法和交替线性化算法进行数值试验得到的数值结果。

2 方法介绍

2.1 混合近似临近点算法

2.1.1 *算法提纲*　本节将讨论混合近似临近点算法。HAPPA 是一种迭代方法，它将产生一个序列$\{x^k\}$，和两个辅助序列$\{x_f^k\}$和$\{x_h^k\}$，k 是迭代次数。此算法的每次迭代包含两个子问题：h-子问题和 f-子问题，以及应用于这两个子问题之后的更新准则。

首先，令 $x_f^0 = x^1$，x^1 是初始点，$s_f^0 = \nabla f(x_f^0)$。

h-子问题：

$$\min_x \widetilde{f}_k(x) + h(x) + \frac{1}{2}\| x - x^k \|_D^2 \tag{3}$$

记 h-子问题的解为 x_h^k。$\widetilde{f}_k(x)$是 $f(x)$在点 x_f^{k-1} 处的线性逼近函数，定义 $\widetilde{f}_k(x) = f(x_f^{k-1}) + <s_f^{k-1}, x - x_f^{k-1}>$。因为(3)式在点 x_h^k 处存在最优的充分必要条件为$\nabla h(x_h^k) + s_f^{k-1} + D(x_h^k - x^k) = 0$，令 $s_h^k = \nabla h(x_h^k)$，则 $s_h^k = -s_f^{k-1} - D(x_h^k - x^k)$。

f-子问题：
$$\min_x f(x)+\tilde{h}_k(x)+\frac{1}{2}\|x-x^{k+1}\|_D^2 \tag{4}$$

记 f-子问题得解为 x_f^k。$\tilde{h}_k(x)$是 $h(x)$在点 x_h^k 的二次逼近函数，定义

$$\tilde{h}_k(x)=h(x_h^k)+<s_h^k,x-x_h^k>+\frac{1}{2!}(x-x_h^k)^{\mathrm{T}}H(x-x_h^k)$$

其中 $H=\begin{bmatrix}\frac{\partial^2 h(x_h^k)}{\partial {x_1}^2} & \cdots & \frac{\partial^2 h(x_h^k)}{\partial x_1 x_N}\\ \vdots & & \vdots \\ \frac{\partial^2 h(x_h^k)}{\partial x_1 x_N} & \cdots & \frac{\partial^2 h(x_h^k)}{\partial {x_N}^2}\end{bmatrix}$。因为(4)式在点 x_f^k 处取得最优的充分必要条件为$\nabla f(x_f^k)+s_h^k+H(x_f^k-x_h^k)+D(x_f^k-x^{k+1})=0$。令 $s_f^k=\nabla f(x_f^k)$，则 $s_f^k=-s_h^k-H(x_f^k-x_h^k)-D(x_f^k-x^{k+1})$。

更新准则：更新准则用于子问题(3)和(4)之后，如果满足更新准则，则它会改变子问题当前的近似解 x^k，更新准则中的参数 $\gamma\in(0,1)$。下面将给出完整的算法步骤。

2.1.2 HAPPA 的基本算法　步骤 0：给定初始点 x^1，$x_f^0=x^1$，$s_f^0=\nabla f(x_f^0)$，$\tilde{f}_1(x)=f(x_f^0)+<s_f^0,x-x_f^0>$，$k=1$。

步骤 1：找出如下 h-子问题的解：$\min\limits_x \tilde{f}_k(x)+h(x)+\frac{1}{2}\|x-x^k\|_D^2$，记解为 x_h^k。

步骤 2：判断 $f(x_h^k)+h(x_h^k)\geqslant f(x_f^{k-1})+h(x_f^{k-1})-\varepsilon$，$\varepsilon$ 为一个微小的正实数，若满足，则停止迭代，跳出循环，否则再判断 $f(x_h^k)+h(x_h^k)\leqslant(1-\gamma)[f(x_f^{k-1})+h(x_f^{k-1})]+\gamma[\tilde{f}_k(x_h^k)+h(x_h^k)]$，$\gamma\in(0,1)$，若满足，则 $x^{k+1}=x_h^k$，转步骤 3，否则 $x^{k+1}=x^k$，转步骤 3。

步骤 3：计算 $s_h^k=-s_f^{k-1}-D(x_h^k-x^k)$，定义$\tilde{h}_k(x)=h(x_h^k)+<s_h^k,x-x_h^k>+\frac{1}{2!}(x-x_h^k)^{\mathrm{T}}H(x-x_h^k)$，找出如下 f-子问题的解：$\min\limits_x f(x)+\tilde{h}_k(x)+\frac{1}{2}\|x-x^{k+1}\|_D^2$，记解为 x_f^k。

步骤 4：判断 $f(x_f^k)+h(x_f^k)\geqslant f(x^{k+1})+h(x^{k+1})-\varepsilon$，若满足，则停止迭代，跳出循环，否则再判断 $f(x_f^k)+h(x_f^k)\leqslant(1-\gamma)[f(x^{k+1})+h(x^{k+1})]+\gamma[f(x_f^k)+\tilde{h}_k(x_f^k)]$，若满足，则 $x^{k+1}=x_f^k$，转步骤 5，否则 $x^{k+1}=x^{k+1}$，转步骤 5。

步骤 5：计算 $s_f^k=-s_h^k-H(x_f^k-x_h^k)-D(x_f^k-x^{k+1})$，定义 $\tilde{f}_{k+1}(x)=f(x_f^k)+<s_f^k,x-x_f^k>$

步骤 6：令 $k=k+1$，回到步骤 1。

下面介绍交替线性化算法(ALM)[6]，便于后面编程进行数值计算与结果比较。

2.2 ALM 的基本算法

步骤 0：选取 $x^1\in\mathrm{dom}\ h$，$z_f^0\in\mathbf{R}^n$，$g_f^0\in\partial f(z_f^0)$，定义 $\tilde{f}_1(x)=f(z_f^0)+<g_f^0,x-z_f^0>$，选取参数 $\rho_1\geqslant\rho_{\min}>0$，$\kappa>1$，$\beta_0>0$，$\beta\in(0,1)$，令 $k=1$。

步骤 1：找出 h-子问题的最优解 z_h^k：$\min\limits_x h(x)+\tilde{f}_k(x)+\frac{1}{2}\rho_k\|x-x^k\|^2$。令 $g_h^k=-g_f^{k-1}-\rho_k(z_h^k-x^k)$，$\tilde{h}_k(x)=h(z_h^k)+<g_h^k,x-z_h^k>$。

步骤 2：下降准则。令 $\tilde{F}_k=h+\tilde{f}_k$，$v_k=F(x^k)-\tilde{F}_k(z_h^k)$，若 $F(z_h^k)\leqslant F(x^k)-\beta_1 v_k$，则令 $x^{k+1}=z_h^k$(下降步)；否则令 $x^{k+1}=x^k$(空步)。

步骤 3：参数 ρ 的更新。若 $x^{k+1}=z_h^k$(下降步)，则 $\rho_{k+1}\in\left[\max\{\rho_{\min},\frac{\rho_k}{\kappa}\},\rho_k\right]$；若 $x^{k+1}=x^k$(空步)，$\delta_k:=F(z_h^k)-\tilde{F}_k(z_h^k)\geqslant\beta_0\frac{v_k}{|z_h^k-x^k|}$，则 $\rho_{k+1}\geqslant\rho_k$；否则 $\rho_{k+1}=\rho_k$。

步骤 4：找出 f-子问题的最优解 z_f^k：$\min\limits_x \tilde{h}_k(x)+f(x)+\frac{1}{2}\rho_{k+1}\|x-x^{k+1}\|^2$。令 $g_f^k=-g_h^k-$

$\rho_{k+1}(z_f^k - x^{k+1})$，$\widetilde{f}_{k+1}(x) = h(z_f^k) + \langle g_f^k, x - z_f^k \rangle$。

步骤 5：令 $k = k+1$，转到步骤 1。

注　文献[6]中没有提及算法的终止准则，笔者在编程时利用的终止准则与算法 2.1.2 的终止准则相同。

3 数值试验

本节给出 3 个数值例子进一步说明本文提出的算法是有效的。数值试验的运行环境：电脑操作系统 Windows XP 专业版 32 位 SP3(DirectX 9.0c)，内存 2 GB，处理器是英特尔 Pentium(奔腾)双核 T4300@2.10 GHz，编程软件 Matlab 6.5。

对于各个算例，分别通过交替线性化算法(记作 ALM)和本文提出的混合近似临近点算法(记作 HAPPA)比较求解，数值结果记录在下面的表格中，其中 x^1 为任意选取的初始点，x^* 代表原问题的最优解，$F(x^*)$代表原问题的最优值，$Iter$ 代表迭代次数，$t(s)$代表 CPU 的运算时间。

例 1　$\min F(x) := h(x) + f(x)$。其中 $f(x) = (x_1 - x_3^2)^2 + (x_2 - x_4^2)^2$，$h(x) = (x_3 - x_2^2)^2 + (x_4 - x_1^2)^2$。

本例摘自文献[13]第 187 页。很容易知道一个最优解为 $x^* = (1,1,1,1)$，最优值为 $F(x^*) = 0$。在 HAPPA 算法中，令 $\varepsilon = 10^{-10}$，$r = 0.5$，$D = diag(100,100,100,100)$。ALM 算法和 HAPPA 算法中，选取不同的初始点解决这个问题，部分数据记录在表 1 中。

表 1　例 1 的两种算法比较

算法	x^1	x^*	$F(x^*)$	$Iter$	$t(s)$
ALM	(12,12,12,12)	(1.002 5,1.002 2,1.002 3,1.002 4)	2.221 8e−005	118	3.250 000
HAPPA	(12,12,12,12)	(1.002 1,1.002 0,1.001 9,1.002 1)	1.662 7e−005	118	3.703 000
ALM	(10,10,10,10)	(1.002 6,1.002 7,1.002 7,1.002 5)	2.790 2e−005	118	3.250 000
HAPPA	(10,10,10,10)	(1.002 0,1.002 0,1.001 8,1.002 0)	1.574 4e−005	119	3.766 000
ALM	(8,10,10,9)	(1.002 6,1.002 9,1.002 9,1.002 7)	3.177 5e−005	117	3.250 000
HAPPA	(8,10,10,9)	(1.002 4,1.002 2,1.002 4,1.002 0)	2.041 4e−005	119	3.812 000
ALM	(7,7,7,7)	(1.002 8,1.002 7,1.002 7,1.002 7)	2.952 7e−005	113	3.062 000
HAPPA	(7,7,7,7)	(1.002 1,1.001 9,1.002 1,1.002 0)	1.650 8e−005	94	3.016 000
ALM	(4,4,4,4)	(1.002 6,1.002 5,1.002 6,1.002 6)	2.633 2e−005	112	2.985 000
HAPPA	(4,4,4,4)	(1.001 9,1.001 9,1.001 9,1.001 7)	1.373 7e−005	119	3.609 000
ALM	(3,3,3,3)	(1.002 2,1.002 1,1.002 0,1.002 3)	1.875 0e−005	117	3.140 000
HAPPA	(3,3,3,3)	(1.002 1,1.002 0,1.001 9,1.002 0)	1.592 5e−005	115	3.500 000
ALM	(4,3,2,1)	(1.002 3,1.002 4,1.002 2,1.002 1)	2.043 6e−005	116	3.062 000
HAPPA	(4,3,2,1)	(1.002 6,1.002 7,1.002 5,1.002 8)	2.843 4e−005	110	3.375 000

试验结果表明，选取 7 个不同的初始点，采用 ALM 和 HAPPA 两种算法计算，得到 7 组不同的结果，和最优解作比较，可知在前 6 组结果中，HAPPA 的计算效果都比 ALM 的计算效果好。

例 2　$\min F(x) := h(x) + f(x)$，其中 $f(x) = e^{-2x}$，$h(x) = e^x$。

通过简单计算可以知道例 2 的全局最优解是 $x^* = \dfrac{\ln 2}{3} \approx 0.231\ 049\ 060\ 186\ 65$，全局最优值是 $F(x^*) = F\left(\dfrac{\ln 2}{3}\right) = f\left(\dfrac{\ln 2}{3}\right) + h\left(\dfrac{\ln 2}{3}\right) = 2^{(-2/3)} + 2^{(1/3)} \approx 1.889\ 881\ 574\ 842\ 31$，在算法中令 $\varepsilon = 10^{-10}$，$r = 0.5$，$D = 100$，利用 ALM 算法和 HAPPA 算法解决这个问题的一些数据记录在表 2 中。

表2 例2的两种算法比较

算法	x^1	x^*	$F(x^*)$	$Iter$	$t(s)$
ALM	10	0.232 132 919 976 41	1.889 883 794 183 10	111	0.593 000
HAPPA	10	0.231 450 929 803 57	1.889 881 880 015 78	115	0.656 000
ALM	8	0.232 381 998 981 15	1.889 884 931 153 30	109	0.594 000
HAPPA	8	0.231 265 390 263 78	1.889 881 663 279 94	116	0.625 000
ALM	6.5	0.233 310 333 508 46	1.889 891 231 209 86	98	0.547 000
HAPPA	6.5	0.231 239 134 254 23	1.889 881 643 115 91	117	0.657 000
ALM	5	0.233 380 559 284 40	1.889 891 840 056 94	98	0.547 000
HAPPA	5	0.231 270 390 115 75	1.889 881 667 414 99	115	0.672 000
ALM	2.5	0.232 215 625 741 83	1.889 884 145 736 05	102	0.547 000
HAPPA	2.5	0.231 361 942 244 71	1.889 881 759 833 32	111	0.578 000

通过对表2观察分析,显然,文中选取了5个不同的初始点,HAPPA的计算效果均比ALM的计算效果好。

例3 $\min F(x):=h(x)+f(x)$,其中 $f(x)=x_1^2+x_2^2$,$h(x)=e^{x_1^2+x_2^2}$。

容易计算得该问题的全局最优解为 $x^*=(0,0)$,最优值为 $F(x^*)=1$。在ALM算法和HAPPA算法中,令 $\varepsilon=10^{-10}$,$r=0.5$,$D=diag(100,100)$,类似前面的例子,选取不同的初始点,得到的结果记录在表3中。

表3 例3的两种算法比较

算法	x^1	x^*	$F(x^*)$	$Iter$	$t(s)$
ALM	(15,15)	(1.722 125 9, 1.722 170 7)	382.684 51	1	0.141 000
HAPPA	(15,15)	(0.183 397 00e−2, 0.197 910 03e−2)	1.000 014 6	78	0.703 000
ALM	(10,10)	(1.658 786 9, 1.658 715 8)	250.908 43	1	0.140 000
HAPPA	(10,10)	(0.145 816 62e−2, 0.194 733 23e−2)	1.000 011 8	82	0.735 000
ALM	(5,10)	(1.024 032 5, 2.048 029 9)	194.493 06	1	0.125 000
HAPPA	(5,10)	(0.106 639 84e−2, 0.194 715 39e−2)	1.000 009 9	82	0.734 000
ALM	(5, 5)	(1.532 809 4, 1.532 814 1)	114.538 91	1	0.125 000
HAPPA	(5, 5)	(0.195 729 53e−2, 0.172 688 24e−2)	1.000 013 6	79	0.718 000
ALM	(4,4)	(0.192 826 68e−2, 0.175 396 92e−2)	1.000 013 6	77	0.828 000
HAPPA	(4,4)	(0.181 940 69e−2, 0.190 022 86e−2)	1.000 013 8	78	0.703 000
ALM	(4,2)	(0.198 415 33e−2, 0.111 783 41e−2)	1.000 010 4	80	0.656 000
HAPPA	(4,2)	(0.195 779 54e−2, 0.794 593 46e−3)	1.000 008 9	85	0.734 000
ALM	(2,2)	(0.193 573 70e−2, 0.194 155 62e−2)	1.000 015 0	74	0.640 000
HAPPA	(2,2)	(0.156 093 35e−2, 0.195 911 70e−2)	1.000 012 5	76	0.703 000

由表3可知,两种算法各选取了7个不同的初始点,前4个初始点离最优点较远,ALM迭代一次结果就溢出,得到的结果明显比HAPPA得到的结果差,最后3组结果作比较,发现有1组结果非常接近,另外2组中HAPPA得到的结果更好。

4 结论

从上面的数值结果可知:对于同一个问题,ALM算法和HAPPA算法的迭代次数和CPU的运算时间相当,而HAPPA得到的计算结果的精度更高,尤其针对非线性程度较高的函数,如例2和例3,因此HAPPA是有意义的;HAPPA的收敛性目前仍在考虑之中。

参考文献：

[1] Martinet B. Regularisation d'inequations variationelles par approximations successives[J]. RAIRO Rech Oper, 1970, 4(R3) :154-158.

[2] Rockafellar R T. Monotone operators and the proximal point algorithm[J]. SIAM J Control Optim, 1976, 14(5): 877-898.

[3] Bertsekas D P. Convex optimization theory[M]. Beijing: Tsinghua University Press, 2011:346-353.

[4] He B, Yuan X, Zhang W. A customized proximal point algorithm for convex minimization with linear constraints [EB/OL]. (05-07-2013). New York: Springer Science+ business Midea.

[5] Eckstein J, Silva P J S. Proximal methods for nonlinear programming: double regularization and inexact subproblems [J]. Comput Optim Appl, 2010, 46(2): 279-304.

[6] Kiwiel K C, Rosa C H R, Uszczynski A. Proximal decomposition via alternating linearization[J]. SIAM J Optim, 1999, 9(3): 668-689.

[7] Koko J. Parallel uzawa method for large-scale minimization of partially separable functions[J]. J Optim Theory Appl, 2013, 158(1): 172-187.

[8] Goldfarb D, Ma S Q, Scheinberg K, Fast alternating linearization methods for minimizing the sum of two convex functions[EB/OL]. (03-24-2012). New York: Springer-Verlag.

[9] Bertsekas D P. Constrainedo optimization and lagrange multiplier methods [M]. New York: Aca-demic Press, 1982.

[10] Hestenes M R. Multiplier and gradient methods[J]. J Optim Theory Appl, 1969, 4(5): 303-320.

[11] Powell M J. A method for nonlinear constraints in minimization problems[M]. London: Academic Press, 1969: 283-298.

[12] Rockafellar R T. Augmented lagrangians and applications of the proximal point algorithm in convex programming [J]. Math Oper Res, 1976, 1(2): 97-116.

[13] Nocedal J, Stephen J. Wright, numerical optimization[M]. 2nd ed. New York: Springer-Verlag, Inc, 2006: 186-189.

Operations Research and Cybernetics

A Hybrid Approximate Proximal Point Algorithm for Minimizing the Sum of Two Convex Function

CHEN Yongmei, BAI Fusheng

(College of Mathematics, Chongqing Normal University, Chongqing 401331, China)

Abstract: In this article, hybrid approximate proximal point algorithm is proposed to minimize the sum of two convex functions. It replaces the optimization problem in the proximal point algorithm by a series of subproblems of minimizing the approximate function to get the optimal solution of optimization problem. In the subproblems, the function with less nonlinearity is replaced by its linear model and the other is replaced by its quadratic model alternately. Under the framework of the proximal point algorithm, we can find the solution of the original problem. Three numerical examples are given to illustrate the effectiveness of the present algorithm.

Key words: convex programming; approximate proximal point method; linear model; quadratic model

(责任编辑 游中胜)