Journal of Computational Mathematics, Vol.21, No.2, 2003, 237–246.

A NONMONOTONE TRUST REGION ALGORITHM FOR NONLINEAR OPTIMIZATION SUBJECT TO GENERAL CONSTRAINTS*[1)]

Hongchao Zhang[†]
(*Department of Mathematics, University of Florida, Gainesville, FL, USA 32611*)

Abstract

In this paper we present a nonmonotone trust region algorithm for general nonlinear constrained optimization problems. The main idea of this paper is to combine Yuan's technique[1] with a nonmonotone method similar to Ke and Han [2]. This new algorithm may not only keep the robust properties of the algorithm given by Yuan, but also have some advantages led by the nonmonotone technique. Under very mild conditions, global convergence for the algorithm is given. Numerical experiments demonstrate the efficiency of the algorithm.

Key words: Nonlinear optimization, Nonmonotone algorithm, Trust region, General constraintss

1. Introduction

In this article, we consider the following nonlinear programming problem :

$$\min_{x\in\Re^n} \quad f(x) \tag{1.1}$$

$$s.t. \quad c_i(x) = 0, \qquad i = 1, 2, \ldots, m_e; \tag{1.2}$$

$$c_i(x) \geq 0, \qquad i = m_e + 1, \ldots, m, \tag{1.3}$$

where $f(x)$ and $c_i(x)$ $(i = 1, \ldots, m)$ are real functions defined in $\Re^n$, at least one of these functions is nonlinear, and $m \geq m_e$ are two non-negative integers.

In recent years, due to its nice results in real calculations, for example see [3], the nonmonotone trust region method has received many successful applications. One of the important reasons is that the nonmonotone method allows the sequence of iterates to follow the bottom of curved valleys (a common occurrence in difficult nonlinear problems) much more loosely. On the other hand, the monotonic reduction at every iteration, namely $f(x_{k+1}) < f(x_k)$, is not the intrinsic property to the convergence of the trust region method. Especially when the merit function is nondifferentiable, the nonsmoothness of the merit function may cause unnecessary reduction of the trust region bound (see [9]), a phenomenon similar to "Maratos effect". One technique to overcome this undesirable effect is the "second order step", but the price paid is that it must compute the value of the constraints at an auxiliary point and solve an additional subproblem. However the nonmonotone technique, like the watchdog technique, is a simple way which is helpful to overcome this difficulty. This is also one of our motivations to use the nonmonotone technique in our algorithm, as the merit function we use is nondifferentiable.

* Received July 10, 2000.

[1)] This work was done when the author was studying in the State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, P. O. Box 2719, Beijing 100080, China .

† Email address: hzhang@math.ufl.edu.

Up to now most nonmonotone trust region algorithms are applied in the unconstrained optimization, for example see [2] [5] [7], except Toint[3], which solves nonlinear optimization problems subject to convex constraints. In addition, Ke and Han [4] proposed a nonmonotone trust region method for equality constrained optimization problems based on a continuously differentiable merit function. In this paper, we try to present a nonmonotone trust region algorithm with a nondifferentiable merit function which can solve the general constrained optimization problems. The numerical results are also reported with this paper.

The paper is organized as follows. We present our algorithm in section 2 and give some preliminary results in section 3. Global convergence analysis of the algorithm are provided in section 4. Numerical results are reported in section 5. Conclusions are given in the last section.

2. The Algorithm

Define the L_∞ exact penalty function associated with (1.1)–(1.3)

$$P_{k,i}(x) = f(x) + \sigma_{k,i}\|c^-(x)\|_\infty, \tag{2.1}$$

where $\sigma_{k,i}$ is a penalty parameter and $c(x) = (c_1(x), \ldots, c_m(x))$, $c^-(x) \in \Re^m$ with

$$c_i^-(x) = c_i(x), \qquad i = 1, 2, \ldots, m_e; \tag{2.2}$$

$$c_i^-(x) = min(c_i(x), 0), \qquad i = m_e + 1, \ldots, m. \tag{2.3}$$

It is easy to see that $\|c^-(x)\|_\infty = 0$ if and only if x is a feasible point of (1.1)–(1.3). And under certain conditions, we can prove that the minimizer of the L_∞ penalty function is also a solution of the original nonlinear programming problem (1.1)–(1.3).

The subproblem we solve in our algorithm has the following form :

$$\min_{d\in\Re^n} \qquad g_k^T d + \frac{1}{2} d^T B_k d + \sigma_{k,i}\|(c_k + A_k^T d)^-\|_\infty = \phi_{k,i}(d) \tag{2.4}$$

$$s.\ t. \qquad \|d\|_\infty \le \Delta_{k,i}, \tag{2.5}$$

where the superscript "-" has the same meaning as (2.2)–(2.3) and $A_k \in \Re^{n\times m}$ is the Jacobi matrix of the constraints. Assume that an inexact solution $s_{k,i}$ of (2.4)–(2.5) is computed such that it satisfies

$$\phi_{k,i}(0) - \phi_{k,i}(s_{k,i}) \ge \tau\, \epsilon_{k,i} \min[\Delta_{k,i}, \epsilon_{k,i}/\|B_k\|_2], \tag{2.6}$$

where, $0 < \tau \le \frac{1}{2}$ is a constant, $\epsilon_{k,i} = \|g_k - A_k\lambda_{k,i}\|_\infty$ and $\lambda_{k,i} \in \Re^m$ is the Lagrange mutipliers at the current point x_k. Now we give our algorithm.

Algorithm 2.1 *(a nonmonotone trust region algorithm)*

Step 0 *Given* $x_0 \in \Re^n$, $\Delta_{0,0} > 0$, $\epsilon \ge 0$, $B_0 \in \Re^{n\times n}$ *symmetric;*
$\sigma_{0,0} > 0$, $\delta_{0,0} > 0$, $\bar{\epsilon} \ge 0$, *integer* $M \ge 0$, $M_0 = M$;
$1 \gg \eta > 0$, $P_{r(0,0)} = P_{0,0}(x_0)$, $k = i = 0$, $m(0) = 0$.

Step 1 *Solve the subproblem (2.4)-(2.5) for* $s_{k,i}$ *and at the same time compute* $\epsilon_{k,i}$.

Step 2 *If* $\epsilon_{k,i} \le \epsilon$ *and* $\|c^-(x_k)\|_\infty \le \epsilon$, *then stop.*
If $pred_{k,i} = \phi_{k,i}(0) - \phi_{k,i}(s_{k,i}) \le \bar{\epsilon}$ *and* $\|c^-(x_k)\|_\infty \le \epsilon$, *then stop.*
If

$$pred_{k,i} < \sigma_{k,i}\, \delta_{k,i} \min[\Delta_{k,i}, \|c_k^-\|_\infty], \tag{2.7}$$

then

$$\sigma_{k,i+1} = 2\,\sigma_{k,i},\ \delta_{k,i+1} = \frac{1}{4}\,\delta_{k,i},\ \Delta_{k,i+1} = \Delta_{k,i},\ i := i + 1, \tag{2.8}$$

go to Step 1;

else *go to Step* 3.