南京航空航天大学

论文集

（二○○○年）

第20册

八院

南京航空航天大学科技部编

二○○一年六月

八　院

○八一系～○八二系

目 录

序号	姓名	职称	单位	论文题目	刊物、会议名称	年、卷、期	类别
615	倪 勤	教授	081	Limited memory quasi-Newton method for large-scale linearly equality-constrained minimization	ACTA Mathematicae Applicatae Sinica	001603	H
616	戴 华	教授	081	Two algorithms for symmetric linear systems with multiple night-hand sides	Numerical Mathematics-A Journal of Chinese Universities	000901	H
617	戴 华	教授	081	Preconditioning block lanczos algorithm for solving symmetric eigenvalue problems	Journal of Computational Mathematics	001804	H
618	戴 华	教授	081	Perturbation analysis of a class of matrix inverse problems	南京航空航天大学学报（英文版）	001702	J
619	戴 华	教授	081	A numerical method for solving inverse eigenvalue problems	RAIRO Mathematical Modelling and Numerical Analysis	993305	H*
620	杨纪龙 叶尔骅	副教授	081	带有不完全信息随机截尾试验下 Weibull 分布参数的 MLE 的相合性及渐近正态性	应用概率统计	001601	H
621	杨纪龙 叶尔骅 刘海锋	副教授	081	恒加试验下随机载尾指数寿命数据的统计分析	兰州大学学报（自然科学版）	003606	H
622	杨瑞芳	讲师	081	一类具有离散谱的对称微分算子	系统科学与数学	000004	H
623	姜海景 宋 琛	讲师	081	W 混沌与伪移位不变集	南京航空航天大学学报	003205	J
624	邱建贤 赵 宁 戴嘉尊	博士	081	A class of large time-step MUSCL schemes	2000 年第四届亚州计算流体力学会议	2000	
625	邱建贤 戴嘉尊	博士	081	一类求解 Hamilton-Jacofi 方程的交错网络差分格式	南京航空航天大学学报	003205	J
626	刘 萍	讲师	081	复数在化有理真分式为部分分式中的应用	文教资料	000008	
627	刘 萍 吴大伟	讲师	081	π-Supersolvable Groups	Journal of Natural Science Nanjing Normal University	000201	J
628	汪晓虹 周传荣 徐庆华	副教授	081	结构分析模型的修正与振型扩充技术	东南大学学报	003002	H
629	古志鸣	副教授	081	关于 3D 和 2D 型 Spin 态的拓扑分类的注记	南京航空航天大学学报	003206	J
630	王春武 戴嘉尊	讲师	081	欧拉方程组的一类高精度 Boltzmann 型差分格式	燃气涡轮试验与研究	001302	
631	王春武 董焕河 戴嘉尊	讲师	081	高精度 ENO 格式的有效实现	南京航空航天大学学报	003201	J

序号	姓名	职称	单位	论文题目	刊物、会议名称	年、卷、期	类别
632	法 伟 罗成林	助教	082	硅团簇结构和碎片行为的紧束缚理论方法	物理学报	004903	H
633	密国柱	副教授	082	大学课程考试的试题设计	南京航空航天大学学报（社会科学版）	000204	
634	赵志敏 姚红兵 王东新	副教授	082	光激励 SmA 对复合材料刚度的影响	机械工程材料	002402	H
635	赵志敏	副教授	082	Study on the Compatibility of SmA and composite Materials by Holographic Interferometry	Chinese Jourual of Lasers	00B906	H
636	孙 欣 赵志敏	副教授	082	用剪切散斑术研究 SmA 对金属材料形变的影响	激光技术	002402	J

LIMITED MEMORY QUASI-NEWTON METHOD FOR LARGE-SCALE LINEARLY EQUALITY-CONSTRAINED MINIMIZATION*

NI QIN (倪 勤)

(*Nanjing University of Aeronautics and Astronautics, Nanjing* 210016, *China*

and

LSEC, Institute of Computational Mathematics, the Acadmey of Mathematics and Systems Sciences, CAS, the Chiense Academy of Sciences, Beijing 100080, *China*)

Abstract

In this paper, a new limited memory quasi-Newton method is proposed and developed for solving large-scale linearly equality-constrained nonlinear programming problems. In every iteration, a linear equation subproblem is solved by using the scaled conjugate gradient method. A truncated solution of the subproblem is determined so that computation is decreased. The technique of limited memory is used to update the approximated inverse Hessian matrix of the Lagrangian function. Hence, the new method is able to handle large dense problems. The convergence of the method is analyzed and numerical results are reported.

Key words. Limeted memory, quasi-Newton method, large-scale problem, linearly equality-constrained optimization

1. Introduction

Consider the following linearly constrained nonlinear programming problem

$$\begin{aligned} &\min\ f(x),\\ &\text{s.t. } Ax = b, \end{aligned} \tag{1.1}$$

where $x \in R^n$, $A \in R^{m\times n}$ and $f \in C^2$. We are interested in the case when n and m are large and when the Hessian matrix of f is difficult to compute or is dense. It is assumed that A is a matrix of full row rank and that the level set $S(x_0) = \{x : f(x) \le f(x_0),\ Ax = b\}$ is nonempty and compact.

In the past few years, there were two kinds of methods for solving the large-scale problem (1.1). For the one kind, problem (1.1) is solved by using matrix factorization and active set (see [1,2]). These methods are only suitable to large sparse problems. In addition,

Received June 11, 1998.

* This research is supported by the National Natural Science Foundation of China, LSEC Of CAS in Beijing and Natural Science Foundation of Jiangsu Province.

these methods require an initial feasible point. The optimality-condition based methods (see [3,4]) belong to the other kind. According to the Kuhn-Tucker optimality condition, problem (1.1) is transformed into an unconstrained minimization problem. An evident drawback is that the objective function in the unconstrained problem includes the gradient of $f(x)$ in (1.1), which makes the problem somewhat complicated. Hence new methods, especially the efficient codes, are demanded for solving large-scale dense and sparse linearly constrained problems.

In this paper, a new method is proposed and developed for solving large-scale problem (1.1), where the Hessian of f is dense or the second derivative is difficult to compute. In every iteration, a linear equation subproblem is solved by using the scaled conjugate gradient method. A truncated solution of the subproblem is determined so that computation is descreased. With the technique of limited memory update, the Hessian matrix of the Lagrangian function is computed and stored by means of some vectors. Hence the new method is able to handle the dense and sparse large problems.

This paper is organized as follows. A limited memory quasi-Newton method is developed in Section 2. The global convergence is proved in Section 3 and some numerical tests are given in Section 4.

2. Algorithm

The algorithm proposed generates a sequence $\{x_k\}_{k=0}^{\infty}$ of iterates of the form

$$x_{k+1} = x_k + \alpha_k d_k,$$

where d_k is a search direction and α_k is determined by a line search along d_k. First the search direction is discussed in the following.

2.1. Search Direction and Truncated Solution

Consider a sequence of quadratic programming subproblems that approximate the local behavior of problem (1.1) at the current iterate x

$$\begin{aligned} \min \quad & \frac{1}{2}d^{\mathrm{T}}Bd + \nabla f(x)^{\mathrm{T}}d, \\ \text{s.t.} \quad & Ad = -(Ax - b), \end{aligned} \tag{2.1}$$

where $d \in R^n$, B is a positive definite approximation of the Hessian matrix of the Lagrangian function

$$L(x, u) = f(x) - u^{\mathrm{T}}(Ax - b) \tag{2.2}$$

with $u \in R^m$, an approximation of the Lagrangian multiplier vector of (1.1).

In order to avoid finding a basis matrix for the null space of A, consider a dual QP subproblem

$$\min \frac{1}{2}u^{\mathrm{T}}AHA^{\mathrm{T}}u + c(x)^{\mathrm{T}}u, \tag{2.3}$$

where $c(x) = Ax - b - AH\nabla f(x)$, $H = B^{-1}$. Because A is of full row rank, H is positive definite, u in (2.3) is determined by solving

$$AHA^{\mathrm{T}}u = -c(x). \tag{2.4}$$

In order to solve the large-scale problem, consider a truncated solution of linear equation (2.4). If u satisfies

$$\|AHA^{\mathrm{T}}u + c(x)\| \le \min\left\{\delta_1, \delta_2\left\|H(A^{\mathrm{T}}u - \nabla f(x))\right\|, \delta_3\right\},$$

then u is called a truncated solution of (2.4). The choice of δ_1, δ_2 and δ_3 will be discussed later. The truncated solution is obtained by solving (2.4) with a scaling conjugate gradient method. Then, an approximated solution of (2.1) is defined as $d = -H(\nabla f(x) - A^{\mathrm{T}}u)$. For a large-scale problem, the technique of limited memory update will be used to compute and store H.

2.2 Limited Memory Update and Merit Function

According to the results in [5], the inverse limited memory BFGS matrix H_k is

$$H_k(l) = H_k^{(0)} + [S_k \ H_K^{(0)}Y_k] \begin{bmatrix} R_k^{-\mathrm{T}}(D_k + Y_k^{\mathrm{T}}H_k^{(0)}Y_k)R_k^{-1} & -R_k^{-\mathrm{T}} \\ -R_k^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_k^{\mathrm{T}} \\ Y_k^{\mathrm{T}}H_k^{(0)}, \end{bmatrix} \tag{2.5}$$

where

$$\begin{aligned} &S_k = [s_{k-l}, \cdots, s_{k-1}], \qquad Y_k = [y_{k-l}, \cdots, y_{k-1}], \\ &s_j = x_{j+1} - x_j, \qquad y_j = \nabla_x L(x_{j+1}, u_j) - \nabla_x L(x_j, u_j), \\ &(R_k)_{i,j} = \begin{cases} s_{k-l-1+i}^{\mathrm{T}} y_{k-l-1+j}, & \text{if } i \le j, \\ 0, & \text{otherwise}, \end{cases} \\ &D_k = \mathrm{diag}\,[s_{k-l}^{\mathrm{T}} y_{k-l}, \cdots, s_{k-1}^{\mathrm{T}} y_{k-1}]. \end{aligned}$$

$H_k^{(0)}$ is a diagonal positive definite initial matrix, l is a given positive integer often less than 10 (see [6]). For the first few iterations, when $k \le l$, we need only to replace l by k in the formulae above.

The merit function is defined by the augmented Lagrangian approach

$$\Phi_r(x, v) = f(x) - \sum_{j=1}^{m} v_j(a_j^{\mathrm{T}}x - b_j) + \frac{1}{2}\sum_{j=1}^{m} r_j(a_j^{\mathrm{T}}x - b_j)^2, \tag{2.6}$$

where r_j is the j-th penalty parameter and v_j is also the approximated Lagrange multiplier, $j = 1, \cdots, m$. $v_1, \cdots, v_m$ are introduced such that the approximation of Lagrange multiplier is efficiently updated.

2.3. Algorithm

A limited memory quasi-Newton algorithm is described below for solving linearly equality-constrained problem.

Algorithm 2.1.

Step 0. Choose some starting values $x_0 \in R^n$, $v_0 \in R^m$, $H_0 \in R^{n\times n}$, a positive definite diagonal matrix, $r_0 \in R^m$, where $r_j^{(0)} \ge 1$, $j = 1, \cdots, m$.

Step 1. Determine a search direction.

1.1) Compute a truncated solution of (2.4) such that u_k satisfies

$$\|AH_kA^{\mathrm{T}}u + c(x_k)\| \le \min\left\{\delta_k^{(1)}, \delta_k^{(2)}\|H_k(A^{\mathrm{T}}u - \nabla f(x_k))\|, \delta_k^{(3)}\right\}, \tag{2.7}$$

where the choice of $\delta_k^{(1)}, \delta_k^{(2)}, \delta_k^{(3)}$ is refered to in the following remark.

1.2) Define a search direction $(d_k, u_k - v_k)$ where

$$d_k = -H_k(\nabla f(x_k) - A^{\mathrm{T}}u_k). \tag{2.8}$$

Step 2. Compute the penalty parameter vector r_{k+1}. Set

$$r_j^{(k+1)} = \max\Big(\sigma_j^{(k)} r_j^{(k)},\ \frac{2m(u_j^{(k)} - v_j^{(k)})^2}{(\nabla f(x_k) - A^{\rm T}u_k)^{\rm T} H_k (\nabla f(x_k) - A^{\rm T}u_k)}\Big), \tag{2.9}$$

$$\sigma_j^{(k)} = \min\Big(1, \frac{k+1}{\sqrt{r_j^{(k)}}}\Big).$$

Step 3. Perform a line search. A steplength α_k is chosen such that

$$\phi_k(\alpha) \le \phi_k(0) + \mu\alpha\phi_k'(0) \tag{2.10}$$

where $\mu \in (0, \frac{1}{2})$, and $\phi_k(\alpha) = \Phi_{r_{k+1}}\big(x_k + \alpha d_k,\ v_k + \alpha(u_k - v_k)\big)$ where $\Phi_r(x, v)$ is defined by (2.6).

Step 4. Set $x_{k+1} = x_k + \alpha_k d_k,\ v_{k+1} = v_k + \alpha_k(u_k - v_k)$. If termination conditions are satisfied, then stop; otherwise, go to Step 5.

Step 5. Compute H_{k+1} by using the inverse limited memory BFGS matrix defined by (2.5). In order to retain $s_k^{\rm T} y_k > 0$, replace s_k by s_k'. Here

$$s_k = x_{k+1} - x_k, \qquad s_k' = \theta x_k + (1-\theta)H_k y_k,$$

$$\theta = \begin{cases} 1 & \text{if } a \ge 0.2b, \\ 0.8b/(b-a), & \text{otherwise,} \end{cases}$$

where $a = s_K^{\rm T} y_k,\ b = y_k^{\rm T} H_k y_k,\ y_k = \nabla_x L(x_{k+1}, u_k) - \nabla_x L(x_k, u_k),\ k = k+1$. Go to Step 1.

Remark. In Step 1.1, $\delta_k^{(1)}$ is introduced such that a relatively exact solution is obtained. $\delta_k^{(2)}\|H_k(A^{\rm T}u - \nabla f(x))\|$ ensures the getting of a high convergence rate (see Theorem 4.4 in Section 3). In the implementation of Algorithm 2.1, $\delta_k^{(1)}$ and $\delta_k^{(2)}$ are chosen as

$$\delta_k^{(1)} = \frac{1}{(k+1)^2}, \qquad \delta_k^{(2)} = \frac{1}{k+1}.$$

In order to obtain the efficient descent property of the search direction, $\delta_k^{(3)}$ is chosen such that

$$\begin{aligned} &\|u_k - v_k\|\delta_k^{(3)} \le \frac{1}{8}\xi_k, \qquad \|Ax_k - b\|\,\|r\|_\infty \delta_k^{(3)} \le \frac{1}{8}\xi_k, \\ &\|Ax_k - b\|\,\|u_k - v_k\|_\infty \delta_k^{(3)} \le \frac{1}{16m}\xi_k^2, \end{aligned} \tag{2.11}$$

where $\xi_k = \big\|(A^{\rm T}u_k - \nabla f(x_k))^{\rm T} H_k (A^{\rm T}u_k - \nabla f(x_k))\big\|$.

3. Convergence Analysis

In this section, the global convergence of Algorithm 2.1 will be discussed. First, the descent property of the algorithm is shown in the following theoem, where the gradient of $\Phi_{r_{k+1}}(x_k, v_k)$ is

$$\nabla\Phi_{r_{k+1}}(x_k, v_k) = \begin{pmatrix} \nabla f(x_k) - A^{\rm T}\big(v_k - R_{k+1}(Ax_k - b)\big) \\ -(Ax_k - b) \end{pmatrix} \tag{3.1}$$

with

$$R_{k+1} = \mathrm{diag}\,(r_1^{(k+1)}, \cdots, r_m^{(k+1)}).$$

Theorem 3.1. Let x_k, u_k, v_k, H_k, d_k and r_k be the given iterates of Algorithm 2.1. Then

$$\nabla\Phi_{r_{k+1}}(x_k, v_k)^{\mathrm{T}} \begin{pmatrix} d_k \\ u_k - v_k \end{pmatrix} \le -\frac{1}{4} d_k^{\mathrm{T}} H_k^{-1} d_k. \tag{3.2}$$

Proof. To simplify the notation, the iteration index k is dropped and $k+1$ is replaced by $+$ in the proof. From Step 1.2 of Algorithm 2.1 we obtain that d and u satisfy

$$H^{-1}d + \nabla f(x) - A^{\mathrm{T}}u = 0. \tag{3.3}$$

Defining $\xi = -(Ad + Ax - b)$, then $Ad = -(\xi + Ax - b)$ and

$$\begin{aligned}
& -\nabla\Phi_r(x, v)^{\mathrm{T}} \begin{pmatrix} d \\ u - v \end{pmatrix} \\
=& -\nabla f(x)^{\mathrm{T}} d + d^{\mathrm{T}} A^{\mathrm{T}} (v - R_+(Ax - b)) + (u - v)^{\mathrm{T}} (Ax - b) \\
=& d^{\mathrm{T}} H^{-1} d - u^{\mathrm{T}} A d + d^{\mathrm{T}} A v - d^{\mathrm{T}} A^{\mathrm{T}} R_+(Ax - b) + (u - v)^{\mathrm{T}} (Ax - b) \qquad \text{from (3.3)} \\
=& d^{\mathrm{T}} H^{-1} d + (Ax - b)^{\mathrm{T}} R_+(Ax - b) + 2(u - v)^{\mathrm{T}} (Ax - b) + (u - v)^{\mathrm{T}} \xi + \xi^{\mathrm{T}} R_+(Ax - b).
\end{aligned}$$

From

$$\begin{aligned}
& (Ax - b)^{\mathrm{T}} R_+(Ax - b) + 2(u - v)^{\mathrm{T}} (Ax - b) \\
=& \left\| R_+^{1/2}(Ax - b) + R_+^{-1/2}(u - v) \right\|^2 - \left\| R_+^{-1/2}(u - v) \right\|^2 \\
\ge& -\left\| R_+^{-1/2}(u - v) \right\|^2 = (u - v)^{\mathrm{T}} R_+^{-1}(u - v) \\
\ge& -\frac{1}{2}(\nabla f(x) - A^{\mathrm{T}}u)^{\mathrm{T}} H (\nabla f(x) - A^{\mathrm{T}}u) = -\frac{1}{2} d^{\mathrm{T}} H^{-1} d,
\end{aligned}$$

it follows that

$$\begin{aligned}
-\nabla\Phi_r(x, v)^{\mathrm{T}} \begin{pmatrix} d \\ u - v \end{pmatrix} &\ge \frac{1}{2} d^{\mathrm{T}} H^{-1} d + (u - v)^{\mathrm{T}} \xi + \xi^{\mathrm{T}} R_+(Ax - b) \\
&\ge \frac{1}{2} d^{\mathrm{T}} H^{-1} d - \|u - v\| \, \|\xi\| - \|Ax - b\| \, \|R_+\| \, \|\xi\|.
\end{aligned} \tag{3.4}$$

From the previous definition of ξ, (2.7) and (2.8) we have

$$\xi = -[Ad + (Ax - b)] = AHA^{\mathrm{T}}u - AH\nabla f(x) - (Ax - b)$$

and $\|\xi\| \le \delta_k^{(3)}$. This inequality, with (2.9) and (2.10), implies that

$$\begin{aligned}
& (\|u - v\| + \|Ax - b\| \, \|R_+\|) \, \|\xi\| \le (\|u - v\| + \|Ax - b\| \, \|R_+\|) \, \delta_k^{(3)} \\
\le& \left(\frac{1}{8} + \frac{1}{8}\right) \|(\nabla f(x) - A^{\mathrm{T}}u)^{\mathrm{T}} H (\nabla f(x) - A^{\mathrm{T}}u)\| = \frac{1}{4} d^{\mathrm{T}} H^{-1} d.
\end{aligned}$$

Hence, because of (3.4),

$$-\nabla\Phi_r(x, v)^{\mathrm{T}} \begin{pmatrix} d \\ u - v \end{pmatrix} \ge \frac{1}{4} d^{\mathrm{T}} H^{-1} d,$$

which is the desired result of the theorem.

In order to prove the global convergence theorem of the algorithm, we require a similar lemma to that in [7].

Lemma 3.2. Let x_k, u_k, v_k, H_k, d_k and r_k be the given iterates of Algorithm 2.1. and assume that

(i) there is a positive constant γ_1 such that $d_k^T H_k^{-1} d_k \geq \gamma_1 \|d_k\|^2$ for all k,

(ii) $\{x_k\}, \{d_k\}, \{u_k\}, \{v_k\}, \{H_k\}$ are bounded.

Then for each $\eta > 0$, there exist a k with $\|d_k\| \leq \eta$ and $\|R_{k+1}^{-1/2}(u_k - v_k)\| \leq \eta$.

The proof of the lemma follows from Theorem 3.1 in this section and the proof of Theorem 4.6 in [7].

Theorem 3.3. Let x_k, u_k, v_k, H_k, d_k and r_k be the given iterates of Algorithm 2.1 and assume that all assumptions of Lemma 3.2 hold. Then either Algorithm 2.1 terminates with a Kuhn-Tucker pair (x_{k+1}, u_k) in a finite number of iterations, or any accumulation point (x^*, u^*) of the sequence $\{(x_{k+1}, u_k)\}$ is a Kuhn-Tucker pair of (1.1). Moreover, if the penalty parameter vector r_k is bounded, then there exists an infinite subset $S \subseteq N$ such that $\lim_{k\in S} v_{k+1} = u^*$, where N is the set of natural numbers.

Proof. Let (x^*, u^*) be an accumulation point of $\{(x_{k+1}, u_k)\}$. From Lemma 3.2 it follows that there exists an infinite subset $S \subseteq N$ such that

$$\begin{aligned} &\lim_{k\in S, k\to\infty} x_k = x^*, \qquad \lim_{k\in S, k\to\infty} u_k = u^*, \\ &\lim_{k\in S, k\to\infty} v_k = v^*, \qquad \lim_{k\in S, k\to\infty} d_k = d^* = 0, \\ &\lim_{k\in S, k\to\infty} \|R_{k+1}^{-1/2}(u_k - v_k)\| = 0. \end{aligned} \tag{3.5}$$

From Step 5 of Algorithm 2.1 we obtain that H_k is positive definite for all k. From (2.8) it follows that $\nabla f(x^*) - A^T u^* = 0$. With (2.7) and $c(x_k) = Ax_k - b - AH_k \nabla f(x_k)$, we obtain $Ax^* = b$. Hence, (x^*, u^*) is a Kuhn-Tucker pair of (1.1).

From Step 0 and Step 2, we have

$$r_j^{(0)} \geq 1, \qquad r_j^{(k+1)} \geq \min\left(r_j^k, (k+1)\sqrt{r_j^k}\right),$$

which implies that $r_j^k \geq 1$, $j = 1, \cdots, m$, $k = 0, 1, 2, \cdots$. If r_k is bounded, it follows from (3.5) that $\lim_{k\in S} v_k = u^*$. The theorem is proved.

In Algorithm 2.1, if H_{k+1} in Step 5 is computed by using the usual BFGS inverse update, i.e.

$$H_{k+1} = H_k + \frac{s_k s_k^T}{s_k^T y_k}\left(1 + \frac{y_k^T H_k y_k}{s_k^T y_k}\right) - \frac{1}{s_k^T y_k}(s_k y_k^T + H_k y_k s_k^T), \qquad k = 0, 1, 2, \cdots, \tag{3.6}$$

then the superlinear convergence of the algorithm is obtained, which is discussed in the following theorem.

Theorem 4.4. Let x_k, u_k, v_k, d_k be the given iterates generated by Algorithm 2.1, and H_k be computed according to (3.6). Assume that

(i) the sequence $\{z_k\}$ converges to z^*, where $z_k = \binom{x_k}{u_k}$,

(ii) the steplength α_k is the one when $k \geq k_0$ for a sufficiently large k_0,

(iii) $\lim_{k\to\infty} \frac{\|\nabla_x L(x_{k+1}, u_k)\|}{\|z_{k+1} - z_k\|} = 0$.

Then

$$\lim_{k\to\infty} \frac{\|E(x_{k+1}, u_k)\|}{\|z_{k+1} - z_k\|} = 0, \tag{3.7}$$

where $E(x, u) = \left[\nabla_x L(x, u), u_1 g_1(x), u_2 g_2(x), \cdots, u_m g_m(x)\right]^T$, $\left(g_1(x), \cdots, g_m(x)\right)^T = Ax - b$.

Proof. According to the definition, we have

$$\|E(x_{k+1}, u_k)\| \le \|\nabla_x L(x_{k+1}, u_k)\| + \sum_{j=1}^{m} |u_i^{(k)} g_i(x_{k+1})|.$$

From (2.7) and (2.8), it follows that for $k \ge k_0$

$$\|g(x_{k+1})\| = \|g(x_k) + Ad_k\| = \|AH_kA^{\mathrm{T}}u_k + c(x_k)\|$$
$$\le \delta_k^{(2)}\|d_k\| = \delta_k^{(2)}\|x_{k+1} - x_k\| = o(\|x_{k+1} - x_k\|),$$

which implies that

$$\|E(x_{k+1}, u_k)\| \le \|\nabla_x L(x_{k+1}, u_k)\| - m\,\|u_k\|_\infty \|g(x_{k+1})\|_\infty$$
$$\le \|\nabla_x L(x_{k+1}, u_k)\| + m\,\|u_k\|_\infty o(\|x_{k+1} - x_k\|).$$

In view of the assumption (iii), (3.7) holds.

The superlinear convergence is easily obtained from Theorem 4.4 and some theorems in [8]. It is noted that Algorithm 2.1 does not possess the property of superlinear convergence. because the limited memory update is used in the algorithm. However, the result in Theorem 4.4 is a main motivation of the choice of the termination criteria in (2.7).

4. Numerical Tests

Some numerical results of Algorithm 2.1 are reported and discussed in this section. Computations are carried out on an SGI Indigo R4000 XS workstation. All codes are written in FORTRAN with double precision. In all runs, we choose $l = 5$ in (2.5) and the following termination condition:

$$\|Ax - b\|_1 \le 10^{-4}, \qquad \|\nabla_x L(x, u)\|_2 \le 10^{-3}. \tag{4.1}$$

First consider the following class of problems:

$$\begin{aligned} \min \quad & \sum_{i=1}^{p} \phi((Hx - c)_i), \\ \text{s.t.} \quad & Ax = b, \end{aligned} \tag{4.2}$$

where $A \in R^{m\times n}$, $H \in R^{p\times n}$, $b \in R^n$, $c \in R^p$, $p = 2n$, $m = n/2$, $\phi(t) = \log(\cosh(t))$ and n is an even number. The mathematical model (4.2) is often used to find robust estimators of the parameters. In a way similar to that in [4], the data of (4.2) are chosen according to the following definition:

a) Let $A = (a_{ij})$, $a_{ij} = (i - j)/n$. $H = (h_{ij})$, $h_{ij} = (i - 3j)/(i + 3j)$.

b) Compute a random "solution" x^* such that $x^* \in (-10, 10)$, $i = 1, 2, \cdots, n$.

c) Let $b = Ax^*$.

d) Let $c^* = Hx^*$ and $c_i = c_i^*(1 + r_i\,\mathrm{pert})$, $i = 1, 2, \cdots, n$, where pert $\in (0, 1)$ represents a percental perturbation on the vector c^*, and r_i is random between -1 and 1.

Algorithm 2.1 is used to solve problem (4.2) where n and pert are chosen as different values. The numerical results are shown in Table 1, where

ITER: Number of iterations.

NF: Number of objective function evaluations.

NDF: Number of gradient evaluations,

VISUM: Sum of constraint violations (see (4.1)),

DLAG: Norm of the gradient of Lagrangian function (see (4.1)).

Table 1. Numerical Results of (4.2)

N	pert	ITER	NF	NDF	VISUM	DLAG
20	0.0	39	80	40	3.40D−9	2.45D-5
	0.1	46	82	47	2.45D−11	3.87D−4
	0.2	51	75	52	7.51D−10	2.41D−5
	0.3	57	91	58	6.87D−9	7.85D−5
	0.4	61	85	62	5.39D−10	1.27D−4
	0.5	65	93	66	4.12D−8	8.97D−4
50	0.0	63	102	64	2.47D−8	7.54D−5
	0.1	71	114	72	3.56D−7	6.39D−5
	0.2	68	109	69	3.48D−6	3.40D−6
	0.3	58	98	59	2.17D−5	1.26D−5
	0.4	78	98	79	3.60D−7	6.42D−4
	0.5	90	130	91	7.40D−7	3.30D−4
120	0.0	112	156	113	5.73D−7	2.36D−4
	0.1	105	147	106	2.30D−6	8.10D−5
	0.2	98	133	99	8.51D−7	3.45D−4
	0.3	132	178	133	3.62D−8	2.37D−4
	0.4	120	180	121	8.15D−7	3.45D−5
	0.5	90	47	91	4.25D−6	8.35D-5

The numerical results show that these problems are successfully solved by Algorithm 2.1. The termination condition is satisfied within 150 iterations.

In order to evaluate the algorithm performance further, consider the following test problem

$$\min f(x) = \frac{1}{n_7}\sum_{j=0}^{n_7-1}(x_{7j+1}-10)^2+5(x_{7j+2}-12)^2+4x_{7j+3}^4$$
$$+3(x_{7j+4}-11)^2+10x_{7j+5}^6+7x_{7j+6}^2+x_{7j+7}^4, \tag{4.3}$$
$$\text{s.t.} \begin{pmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & A_{n_7} \end{pmatrix} x = b,$$

where $A_j \in R^{3\times 7}$, $j=1,2,\cdots,n_7$, $A_1=A_2=\cdots=A_{n_7}$ and $A_1=(a_{ik}), a_{ik}=i-k$, $i=1,2,3$; $k=1,\cdots,7$, b is generated in the same way as above. $f(x)$ in (4.3) is a modification of the objective function of the 100-th problem in [7]. Numerical results of (4.3) are shown in Table 2.

Table 2. Numerical Results of (4.3)

N	M	ITER	NF	NDF	VISUM	DLAG
14	6	25	56	26	1.37D−6	3.62D−5
28	12	34	73	35	3.58D−7	3.74D−5
56	24	49	95	50	6.82D−6	5.33D−4
112	48	120	190	121	7.10D−6	4.82D−5
224	96	98	203	99	3.58D−6	5.22D−4

In Table 2, the number of variables n is chosen from 14 to 224, and the number of constraints m from 6 to 96. The results show that these test problems are also successfully solved by Algorithm 2.1.

These results indicate that Algorithm 2.1 can handle medium-scale linearly equality-constrained problems. Hence, it can be believed that with further development, Algorithm 2.1 is capable of processing large-scale linearly constrained problems. In addition, Algorithm 2.1 will be extended to solving large-scale linearly equality and inequality constrained problems. In this case, $A^T HA$ in the subproblem is in general not positive definite. This will be further researched.

References

1 A. Forsgren, W. Murry. Newton Methods for Large-scale Linear Equality-constrained Minimization. *SIAM J. Matrix Anal. Appl.*, 1993, 14: 560–587

2 A. Forsgren, W. Murry. Newton Methods for Large-scale Linear Inequality-constrained Minimization. *SIAM J. Optimization*, 1997, 7: 162–176

3 C. Kanzow. An Unconstrained Optimization Technique for Large-scale Linearly Constrained Convex Minimization Problems. *Computing*, 1994, 53: 101–117

4 A. Friedlander, J.M. Martinez, A. Santos. On the Solution of Linearly Constrained Convex Minimization Problems. *SIAM J. Optimization*, 1994, 4: 331–339

5 R.H. Byrd, J. Nocedal, R.B. Schnabel. Representations of Quasi-Newton Matrices and their Use in Limited Memory Methods. *Mathematical Programming*, 1994, 63(4): 129–156

6 Q. Ni, Y. Yuan. A Subspace Limited Memory Quasi-Newton Algorithm for Solving Large-scale Nonlinear Bound Constrained Optimization. *Mathematics of Computation*, 1997, 66: 1509–1520

7 K. Schittkowski. On the Convergence of a Sequential Quadratic Programming Method with an Augmented Lagrangian Line Search Function. *Math. Operationsforch. u. Statist.* (Seriers Optimization), 1983, 14: 197–216

8 S.P. Han. Superlinear Convergent Variable Metric Algorithm for General Nonlinear Programming Problems. *Math. Prog.*, 1976, 11: 263–282

9 W. Hock, K. Schittkowski. Test Examples for Nonlinear Programming Codes. Lecture Notes in Economics and Mathematical Systems 187, Springer, Berlin, Heidelberg, 1981

English Series | NUMERICAL MATHEMATICS A Journal of Chinese Universities | Vol. 9, No. 1 May 2000

TWO ALGORITHMS FOR SYMMETRIC LINEAR SYSTEMS WITH MULTIPLE RIGHT—HAND SIDES*

Dai Hua(戴华)

Abstract *In this paper, we investigate the block Lanczos algorithm for solving large sparse symmetric linear systems with multiple right—hand sides, and show how to incorporate deflation to drop converged linear systems using a natural convergence criterion, and present an adaptive block Lanczos algorithm. We propose also a block version of Paige and Saunders' MINRES method for iterative solution of symmetric linear systems, and describe important implementation details. We establish a relationship between the block Lanczos algorithm and block MINRES algorithm, and compare the numerical performance of the Lanczos algorithm and MINRES method for symmetric linear systems applied to a sequence of right-hand sides with that of the block Lanczos algorithm and block MINRES algorithm for multiple linear systems simultaneously.*

Key words *Symmetric matrices, multiple linear systems, block Lanczos algorithm, block MINRES method.*

AMS (1991)subject classifications *65F10, 65F15.*

1 Introduction

In many applications we need to solve multiple systems of linear equations

$$Ax^{(i)} = b^{(i)},\ i = 1, \cdots, s \tag{1}$$

with the same $n \times n$ real symmetric coefficient matrix A, but s different right-hand sides $b^{(i)} (i=1, \cdots, s)$. If all of the right-hand sides are available simultaneously, then these s

* The research was supported by the National Natural Science Foundation of China, the Jiangsu Province Natural Science Foundation, the Jiangsu Province "333 Engineering" Foundation and the Jiangsu Province "Qinglan Engineering" Foundation. This work was done while the author was visiting CERFACS, France in March-August 1998.

Received: Oct. 27, 1998.

linear systems can be summarized in block form as follows.

$$AX = B \tag{2}$$

where $X=[x^{(1)}, \cdots, x^{(s)}]$ and $B=[b^{(1)}, \cdots, b^{(s)}]$. If the order n of the matrix A is small, we can solve (2) using direct methods, first decompose the coefficient matrix A into simpler factors at a cost of $O(n^3)$ operations and then solve for each right-hand side at a cost of $O(n^2)$. Direct methods can also be advantageous if the right-hand sides are not all simultaneously available. However, for n large direct methods can be prohibitively expensive both in terms of memory and computational cost, and so iterative methods become appealing. We will assume that the order n of A is sufficiently large and it is possible to keep only a limited number of dimension$-n$ vectors in memory and all of the right-hand sides are available simultaneously and $s \ll n$.

Most of iterative methods are tailored to the solution of linear systems with a single right-hand side. They could be trivially used to solve multiple right-hand problems (2) by simply solving the s linear systems (1) individually. The Lanczos algorithms proposed by Parlett[15] and extended by Saad[16], van der Vorst[21], Papadrakakis and Smerou[14] are mostly suitable when all $b^{(i)}$ are not simultaneously available. Reference[14] is of particular interest as it contains numerical experiments from actual applications. However, it can be significantly more efficient to apply block variants of the iterative methods that generate iterates for all the multiple linear systems simultaneously.

In the block approach, one extends an iterative method for single to multiple systems, by devising a block version which is applied to the block formulation (2) of multiple linear systems (1). The approximate solutions that are generated by a block method for the s different right-hand sides will generally converge at different stages of the block iteration. An efficient block method needs to be able to detect and adaptively deflate converged systems. O'Leary[11] was the first to devise the block conjugate gradient method and the block Lanczos algorithm for multiple symmetric linear systems, which are closely related to the conjugate gradient method[8], the classical Lanczos method[9] and the block Lanczos process [3, 7]. However, the algorithms in [11, 12] can not handle deflations or variable block sizes. For multiple symmetric positive definite systems, Sadkane and Vital[18] proposed the block Davidson method, Calvetti and Reichel [1] devised an adaptive Richardson iteration method. Nikishin and Yeremin [10] presented also a block version of the conjugate gradient algorithm that allows varying block sizes. The literature for nonsymmetric linear systems with multiple right-hand sides is vast[17]. Some methods that have been proposed are block generalizations of solvers for nonsymmetric linear systems: the block biconjugate gradient algorithm[11, 20], block GMRES[22, 2], hybrid GMRES [19] and block QMR method[6]. Although the block methods for multiple nonsymmetric systems could be directly used to solve multiple symmetric systems, they can not make full use of the symmetry of the matrix A.

In this paper, we investigate the block Lanczos algorithm (referred to as BLanczos hereafter) for multiple symmetric linear systems (2), and consider its deflation procedure. Using a natural convergence criterion, we can identify and drop linear systems whose solutions can be recovered from the solution of the remaining multiple linear systems. In order to smooth the possibly erratic behavior of the residual norm curve generated by the application of the BLanczos algorithm to multiple systems (2), we describe a block version of Paige and Saunders' MINRES method[13] for the iterative solution of symmetric linear systems with a single right-hand side. The MINRES iterates are defined by a minimization of the residual norm, which leads to smooth convergence behavior. The block MINRES algorithm (referred to as BMINRES hereafter) is an extension of MINRES to multiple symmetric linear systems. The BMINRES iterates are then determined via a block smoothing residual technique, which can be formulated as a matrix least-squares problem. In order to deflate the converged block iterates, we restart the algorithm with new full rank bases.

The structure of the paper is as follows. In section 2 we briefly review the block Lanczos process, and then discuss the deflation of the BLanczos algorithm for multiple symmetric linear systems, and present an adaptive BLanczos algorithm which allows varying block sizes. In section 3 we describe the BMINRES algorithm and show how it is affected by deflation, and give some implementation details for the BMINRES algorithm. In section 4 we discuss the relation between the BLanczos and the BMINRES algorithms. In section 5 we compare the numerical performance of the Lanczos algorithm and MINRES method for symmetric linear systems applied to a sequence of right-hand sides with that of the BLanczos and BMINRES algorithms for multiple linear systems simultaneously and give results of numerical experiments. Finally, conclusions are presented in section 6.

The following notation will be used. $\|\cdot\|_2$ denotes the Euclidean vector norm or induced spectral norm, and $\|\cdot\|_F$ the Frobenius matrix norm. I_n is the identity matrix of order n. $E_i=[0, \cdots, 0, I_s, 0, \cdots, 0]^T$ with I_s at the ith block position; the total size of E_i will be made clear from the context.

2 The block Lanczos algorithm

We briefly review the block Lanczos process. For $R_0 \in \mathbb{R}^{n\times s}$, let $R_0=Q_1T_1$ be a QR factorization of R_0, i. e. , $Q_1 \in \mathbb{R}^{n\times s}$ is orthonormal and $T_1 \in \mathbb{R}^{s\times s}$ is upper-triangular. The block Lanczos process generates a sequence of orthogonal mutually matrices $Q_k \in \mathbb{R}^{n\times s}$, $k=1, 2, \cdots$, i. e. ,

$$Q_k^T Q_j = \begin{cases} I, & j=k \\ 0, & j\neq k \end{cases}$$

that satisfy a three-term recursion relation

$$Q_{k+1}T_{k+1} = AQ_k - Q_kM_k - Q_{k-1}T_k^T,\ k \geqslant 1 \tag{3}$$

where $Q_0=0$, and M_k, $T_k \in \mathbb{R}^{s\times s}$. Define

$$\mathrm{K}_k(A,\ R_0) = \mathrm{span}\{R_0,\ AR_0,\ \cdots,\ A^{k-1}R_0\} \tag{4}$$

$\mathrm{K}_k(A,\ R_0)$ is referred to as the kth block Krylov subspace. The block Lanczos process constructs an orthonormal basis for the block Krylov subspace $\mathrm{K}_k(A,\ R_0)$. This can be accomplished by using the following algorithm.

Algorithm 2.1 (Block Lanczos process)

1). Compute the QR factorization of R_0:

$$R_0 = Q_1T_1$$

where $Q_1 \in \mathbb{R}^{n\times s}$ is orthonormal and $T_1 \in \mathbb{R}^{s\times s}$ is upper-triangular, and

$$M_1 = Q_1^TAQ_1$$

2). For $j=1,\ \cdots,\ k$, do

2.1). Compute $P_{j+1}=AQ_j-Q_jM_j-Q_{j-1}T_j^T\ (Q_0=0)$

2.2). Compute QR factorization of P_{j+1}:

$$P_{j+1} = Q_{j+1}T_{j+1}$$

where $Q_{j+1} \in \mathbb{R}^{n\times s}$ is orthonormal and $T_{j+1} \in \mathbb{R}^{s\times s}$ is upper-triangular.

2.3). Compute

$$M_{j+1} = Q_{j+1}^TAQ_{j+1}$$

end do.

The matrices M_j, T_j are uniquely determined if the diagonal elements of the T_j generated by Algorithm 2.1 are positive. We now assume this to be the case. We will consider the case of a rank deficiency of P_j later. Let

$$\hat{Q}_k = [Q_1,\ \cdots,\ Q_k] \tag{5}$$

$$\hat{T}_k = \begin{pmatrix} M_1 & T_2^T & & & & \\ T_2 & M_2 & T_3^T & & & \\ & T_3 & \cdot & \cdot & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & T_k^T \\ & & & & T_k & M_k \end{pmatrix} \tag{6}$$

The three-term recursion relation (3) can be written in matrix form as

$$A\hat{Q}_k = \hat{Q}_k\hat{T}_k + P_{k+1}E_k^T = \hat{Q}_k\hat{T}_k + Q_{k+1}T_{k+1}E_k^T \tag{7}$$

where $\hat{Q}_k \in \mathbb{R}^{n\times ks}$ is orthonormal, $\hat{T}_k = \hat{Q}_k^T A \hat{Q}_k \in \mathbb{R}^{ks\times ks}$ is an orthogonal projection of A onto the block Krylov subspace $\mathrm{K}_k(A,\ R_0)$, and is a symmetric band matrix with the band width $2s+1$.

In the BLanczos algorithm the goal is to solve the multiple symmetric linear systems (2). Giving a block of initial guesses $x_0^{(i)}\ (i=1,\cdots,s)$, we define R_0 the block of initial