Gradient Methods for Large Scale Convex Quadratic Functions*

Ya-xiang Yuan
State Key Laboratory of Scientific/Engineering Computing,
Institute of Computational Mathematics and Scientific/Engineering Computing,
Academy of Mathematics and Systems Science, Chinese Academy of Sciences,
Zhong Guan Cun Donglu 55, Beijing, 100190, P.R. China,
E-mail: yyx@lsec.cc.ac.cn

1 Introduction

In this chapter we consider the gradient methods for minimizing large scale convex quadratic functions. Most inverse problems can be formulated as

$$\mathcal{L}x = z, \tag{1.1}$$

where z is the given data or observations, and $\mathcal{L}$ is an mapping. x is the unknown that needs to computed. After discretization and linearization, we will need to solve a set of linear equations

$$Ax = b \quad x \in \Re^n, \tag{1.2}$$

where $b \in \Re^n$ and $A \in \Re^{n\times n}$ is a symmetric positive definite matrix. Many inverse problems can be formulated into (1.4) with a very large n and A is ill-conditioned (for example, see[16, 17, 18, 19]). It is easy to see that linear system (1.2) is equivalent to the following unconstrained optimization problem:

$$\min_{x\in R^n} f(x), \tag{1.3}$$

with

$$f(x) = \frac{1}{2}x^T Ax - b^T x. \tag{1.4}$$

The gradient method is one of the most simple methods for solving (1.3) where $f(x)$ is a continuously differentiable function in $\Re^n$. Assume that $g(x) = \nabla f(x)$ can be obtained at every x. Given an iterate point x_k, the gradient method chooses the next iterate point x_{k+1} in the following form:

$$x_{k+1} = x_k - \alpha_k g_k, \tag{1.5}$$

where $g_k = g(x_k)$ is the gradient at x_k and $\alpha_k > 0$ is a step-length. The gradient method has the advantages of easy to program and suitable for large scale problems. Different step-lengths α_k give different gradient algorithms. If $\alpha_k = \alpha_k^*$ where α_k^* satisfies

$$f(x_k - \alpha_k^* g_k) = \min_{\alpha>0} f(x_k - \alpha g_k), \tag{1.6}$$

*this work is partially supported by Chinese NSF grant 10831006 and by CAS grant kjcx-yw-s7 the gradient method is the steepest descent method, which is also called the Cauchy's method. However, the steepest descent method, though it use the "best" direction and the "best" step-length, turns out to be a very bad method as it normally converges very slowly, particularly for ill-conditioned problems.

In this chapter, we discuss the gradient method for the special case when $f(x)$ is a strictly convex quadratic function (1.4) because this special problem appears in inverse problems very often.

We denote the eigenvalues of A by $\lambda_i (i = 1, 2, ..., n)$ and assume that

$$0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n. \tag{1.7}$$

The steepest descent method, which uses the exact line search step

$$\alpha_k^* = \frac{g_k^T g_k}{g_k^T A g_k} = \frac{\|Ax_k - b\|_2^2}{(Ax_k - b)^T A (Ax_k - b)}, \tag{1.8}$$

turns out to converge very slowly when A is ill-conditioned in the sense that the ratio of λ_1/λ_n is very small.

In 1988, Barzilai and Borwein[1] gave two interesting choices for the step-length α_k:

$$\alpha_k^{BB1} = \frac{\|s_{k-1}\|_2^2}{s_{k-1}^T y_{k-1}}, \tag{1.9}$$

$$\alpha_k^{BB2} = \frac{s_{k-1}^T y_{k-1}}{\|y_{k-1}\|_2^2}, \tag{1.10}$$

where

$$s_{k-1} = x_k - x_{k-1} \quad y_{k-1} = \nabla f(x_k) - \nabla f(x_{k-1}). \tag{1.11}$$

Barzilai and Borwein[1] establishes superlinearly convergence results for two dimensional convex quadratic problems. Moreover, numerical results indicate that for convex quadratic funtions $f(x)$ the BB method performs much better than the steepest descent method. Barzilai and Borwein's work triggered many researches on the gradient method in the past two decades. For example, see Dai[2], Dai and Fletcher[3], Dai and Liao[4], Dai et al.[5], Dai and Yuan[6, 7], Dai and Zhang[8], Fletcher[9], Friedlander et al.[10], Nocedal et al.[11], Raydan[12, 13], Raydan and Svaiter[14], Vrahatis et al.[15], Yuan[20, 21] and Zhou et. al.[22].

In this chapter we consider how the the BB method can be further improved. First we generalize the global convergence result on the gradient method with retards, which enable us to use a wider range of step-lengths for the gradient method. Then, we propose to use short BB step-lengths. Numerical results on random generated problems indicate that short BB step-lengths will give improvement over the standard BB step-length, particularly for large-scale and ill-conditioned problems.

2 A Generalized Convergence Result

Barzilai and Borwein[1] proved the superlinearly convergence of their method for the case having only two variables. Raydan[12] established the global convergence of the gradient method with BB step-lengths for general strictly convex quadratic functions. The following general global convergence result was established by Friedlander et. al.[10]:

Theorem 2.1. *Let $f(x)$ be given by (1.4) and A is positive definite. Let m be an positive integer and $q_j \geq 1 (j = 1, 2, ..., m)$ be m positive numbers. Let $\{x_k\}$ be generated by the gradient method (1.5) with the step-length α_k given by*

$$\alpha_k = \frac{(x_{v(k)} - x^*)A^{(\rho(k)-1)}(x_{v(k)} - x^*)}{(x_{v(k)} - x^*)A^{\rho(k)}(x_{v(k)} - x^*)}, \tag{2.1}$$

where $x^* = -A^{-1}g$, $\rho(k) \in \{q_1, q_2, ..., q_m\}$ and $v(k) \in \{k, k-1, ..., \max\{0, k-m\}\}$ for $k = 0, 1, 2, ...$ Then either $x_k = x^*$ for some finite k or the sequence $\{x_k\}$ converges to x^*.

For a proof of the above theorem, please see [10]. We can easily generalize the above theorem to the following more general form:

Theorem 2.2. *Let $f(x)$ be given by (1.4) and A is positive definite. Let m be a positive integer and γ be a positive number. Let $\{x_k\}$ be generated by the gradient method (1.5) with the step-length α_k satisfying*

$$\alpha_k \in \left[\min_{\substack{|\rho| \leq \gamma, \\ \max[0, k-m] \leq j \leq k}} \frac{g_j A^{(\rho-1)} g_j}{g_j A^{\rho} g_j}, \quad \max_{\substack{|\rho| \leq \gamma, \\ \max[0, k-m] \leq j \leq k}} \frac{g_j A^{(\rho-1)} g_j}{g_j A^{\rho} g_j} \right]. \tag{2.2}$$

Then either $x_k = x^*$ for some finite k or the sequence $\{x_k\}$ converges to x^*.

Proof. Our proof is similar to that of Theorem 2.1 which is given in Friedlander et. al.[10].

First it is easy to see that (2.2) implies that

$$0 < \frac{1}{\lambda_n} \leq \alpha_k \leq \frac{1}{\lambda_1} \tag{2.3}$$

holds for all k.

Let the orthogonal decomposition of A be as follows:

$$A = Q\Lambda Q^T, \tag{2.4}$$

where $Q = [q_1, q_2, ..., q_n]$ is an orthogonal matrix and $\Lambda = Diag[\lambda_1, \lambda_2, ..., \lambda_n]$.

For any given initial point x_0, the gradient $g_0 = Ax_0 + g$ can be expressed by

$$g_0 = \sum_{i=1}^{n} \beta_i^{(0)} q_i, \tag{2.5}$$

where $\beta_i^{(0)} \in \Re (i = 1, 2, ..., n)$. Let $g_k = \sum_{i=1}^{n} \beta_i^{(k)} q_i$ for all $k \geq 0$. It follows from (1.5), (1.4), (2.4) and (2.5) that

$$\beta_i^{(k+1)} = (1 - \alpha_k \lambda_i)\beta_i^{(k)} = \prod_{j=0}^{k} (1 - \alpha_j \lambda_i)\beta_i^{(0)}. \tag{2.6}$$

From (2.6), we have that

$$|\beta_1^{(k)}| = |(1 - \alpha_{k-1}\lambda_1)\beta_1^{(k-1)}| \leq \left| \left(1 - \frac{\lambda_1}{\lambda_n}\right) \beta_1^{(k-1)} \right| \leq (1 - \lambda_1/\lambda_n)^k |\beta_1^{(0)}|. \tag{2.7}$$

The above inequality shows that

$$\lim_{k\to\infty} \beta_1^{(k)} = 0\,. \tag{2.8}$$

We see that the theorem is true if we can prove that

$$\lim_{k\to\infty} \beta_i^{(k)} = 0\,, \tag{2.9}$$

for all $i = 1, 2, ..., n$. If this were not true, there exist a positive number $\hat{\delta}$ and an integer $l \in [1, n-1]$ such that (2.9) holds for all $i = 1, ..., l$ and

$$\limsup_{k\to\infty} |\beta_{l+1}^{(k)}| > \hat{\delta} > 0\,. \tag{2.10}$$

For any given positive number δ, we have that

$$\begin{aligned}
\lim_{|\beta_{l+1}^{(k)}|\geq\delta, k\to\infty} \max_{\substack{|\rho|\leq\gamma,\\ k-m\leq j\leq k}} \frac{g_j^T A^{\rho-1} g_j}{g_j^T A^\rho g_j} &= \lim_{|\beta_{l+1}^{(k)}|\geq\delta, k\to\infty} \max_{\substack{|\rho|\leq\gamma,\\ k-m\leq j\leq k}} \frac{\sum_{i=1}^n (\beta_i^{(j)})^2 \lambda_i^{\rho-1}}{\sum_{i=1}^n (\beta_i^{(j)})^2 \lambda_i^{\rho}} \\
&\leq \lim_{|\beta_{l+1}^{(k)}|\geq\delta, k\to\infty} \max_{\substack{|\rho|\leq\gamma,\\ k-m\leq j\leq k}} \frac{\sum_{i=1}^{l+1} (\beta_i^{(j)})^2 \lambda_i^{\rho-1}}{\sum_{i=1}^{l+1} (\beta_i^{(j)})^2 \lambda_i^{\rho}} \\
&= \frac{1}{\lambda_{l+1}}\,,
\end{aligned} \tag{2.11}$$

due to the fact that (2.9) holds for $i = 1, ..., l$ and that inequality $|\beta_{l+1}^{(k)}| \geq \delta$ and relation (2.6) imply $|\beta_{l+1}^{(j)}| \geq \left(\frac{\lambda_1}{\lambda_{l+1}}\right)^m \delta$ holds for all $j \in [\max[0, k-m], k]$. Therefore, there exists a sufficiently large integer $\hat{k}$ such that

$$\alpha_k \leq \frac{11}{10} \frac{1}{\lambda_{l+1}} \tag{2.12}$$

for all k satisfying $k \geq \hat{k}$ and

$$|\beta_{l+1}^{(k)}| \geq \frac{\lambda_1}{\lambda_{l+1}} \frac{\hat{\delta}}{2}\,. \tag{2.13}$$

Thus, for any $k \geq \hat{k}$, if (2.13) holds, we have that

$$\begin{aligned}
|\beta_{l+1}^{(k+1)}| &= |(1 - \alpha_k \lambda_{l+1})||\beta_{l+1}^{(k)}| \\
&\leq \max\left[1 - \frac{\lambda_{l+1}}{\lambda_n}, \quad \left|\lambda_{l+1} \frac{11}{10} \frac{1}{\lambda_{l+1}} - 1\right|\right] |\beta_{l+1}^{(k)}| \\
&\leq \max[1 - \lambda_{l+1}/\lambda_n, \quad 0.1]|\beta_{l+1}^{(k)}|.
\end{aligned} \tag{2.14}$$

On the other hand, if (2.13) fails, from (2.6) we can show that

$$|\beta_{l+1}^{(k+1)}| \leq \max[1 - \frac{\lambda_{l+1}}{\lambda_n}, \lambda_{l+1}/\lambda_1 - 1]|\beta_{l+1}^{(k)}| \leq \lambda_{l+1}/\lambda_1 |\beta_{l+1}^{(k)}| \leq \frac{\hat{\delta}}{2}. \tag{2.15}$$

It follows from (2.14) and (2.15) that

$$\limsup_{k\to\infty} |\beta_{l+1}^{(k)}| \leq \frac{\hat{\delta}}{2} \tag{2.16}$$

which contradicts to (2.10). This completes our proof. $\square$

It should be pointed that the above result can also deduced from a more general convergence result of Dai[2] by showing that Property (A) of Dai[2] holds. From Dai's results it can be shown that the gradient method with (2.2) converges R-linearly. The reason for giving our direct and simple proof is to avoid unnecessary lengthly analysis.

Though the generalization from (2.1) to (2.2) is very simple and straightforward, it does contain more choices for the step-lengths. For example, we can let α_k be the mean values of any two Raleigh ratios:

$$\alpha_k = \frac{1}{2}\left[\frac{g_{j_1}^T A^{\rho_1 - 1} g_{j_1}}{g_{j_1}^T A^{\rho_1} g_{j_1}} + \frac{g_{j_2}^T A^{\rho_2 - 1} g_{j_2}}{g_{j_2}^T A^{\rho_2} g_{j_2}}\right], \tag{2.17}$$

or

$$\alpha_k = \sqrt{\frac{g_{j_1}^T A^{\rho_1 - 1} g_{j_1}}{g_{j_1}^T A^{\rho_1} g_{j_1}} \frac{g_{j_2}^T A^{\rho_2 - 1} g_{j_2}}{g_{j_2}^T A^{\rho_2} g_{j_2}}}. \tag{2.18}$$

Moreover, all the known choices of α_k (see (4.27)-(4.29) of Dai[2]) having Property (A) satisfy our simple condition (2.2).

3 Short BB Steps

When we apply the gradient method to large scale problems, the most important issue is which step-length will give a fast convergence rate. Therefore it is vital important to find what choices of α_k in the interval (2.2) require less number of iterations to reduce the gradient norm to a given tolerance. Much work has been done on this issue. And it seems that up to now the best choice is the adaptive BB step given by Zhou et. al.[22] in which

$$\alpha_k = \alpha_k^{ABB} = \begin{cases} \alpha_k^{BB2}, & \text{if } \alpha_k^{BB2}/\alpha_k^{BB1} < \kappa; \\ \alpha_k^{BB1}, & \text{otherwise,} \end{cases} \tag{3.1}$$

and $\kappa \in (0, 1)$ is a parameter.

The motivation of the ABB step and its derivation can be found in Zhou et. al.[22]. Basically, The ABB step is a hybrid combination of BB1 and BB2 steps, which mainly use the BB1 step unless α_k^{BB2} is much smaller than ^{BB1}.

It is trivial that $\alpha_k^{BB1} \geq \alpha_k^{BB2}$, namely the BB1 step is normally longer than the BB2 step. Numerical results favor the BB1 (the longer BB step). However, to the author's knowledge, there are no sound theoretical results which ensure that BB1 is better than BB2, though most papers choose to study the BB1 step when the BB method is studied. It seems all the theoretical results hold for the BB1 method are also true for the BB2 method. Thus it is very interesting to know why BB1 is better than BB2. It would be nice to establish sound theoretical results to shed light on this question. Unfortunately, we have not yet been able to do so. In the following paragraph, we give an intuitive analysis on the impact of the step-lengths for the gradient method.

In order to obtain a fast convergence, we need to make all the terms

$$\beta_i^{(k+1)} = (1 - \alpha_k \lambda_i)\beta_i^{(k)} = \prod_{j=0}^{k}(1 - \alpha_j \lambda_i)\beta_i^{(0)} \quad (i = 1, 2, ..., n), \tag{3.2}$$

converge to zero as fast as possible. Due to the relation (2.6), remembering that we have $1/\lambda_n \le \alpha_k \le 1/\lambda_1$, we can see that a smaller α_k will reduce $\beta_n^{(k+1)}$ more quickly, while a larger α_k will reduce the other $\beta_i^{(k+1)}$ (particularly with smaller i) more quickly. This observation tells us that either the longer BB step (BB1) or the shorter BB step (BB2) has its own advantage. Theoretically speaking, from the proof of Theorem 2.2, it is more easy to have $\beta_i^{(k)}$ converging to zero for small i (for example, $|\beta_1^{(k+1)}| \le (1-\lambda_1/\lambda_n)|\beta_1^{(k)}|$ for all k). Thus, we should put more weight for reducing $\beta_i(k)$ for large i, which means that we would prefer to use a shorter step-length. This seems to contradict the fact that the larger step (BB1) is better than the smaller step (BB2) based on many numerical tests.

For most test examples, we choose the starting point randomly, which would implies that $|q_i^T(x_0 - x^*)| (i = 1, 2, ..., n)$ are more or less of the same magnitude. Thus, because

$$\beta_i^{(0)} = \lambda_i q_i^T(x_0 - x^*), \quad i = 1, 2, ..., n, \tag{3.3}$$

we would have that $|\beta_n^{(0)}|$ is much larger than the other $|\beta_i^{(0)}|$ if we assume

$$\lambda_n >> \lambda_{n-1}. \tag{3.4}$$

In this case, we will see that the first iteration (with the exact line search) would give a very small step-length $\alpha_0 \approx 1/\lambda_n$. Consequently, $\beta_n^{(1)} \approx 0$ while $\beta_i^{(1)} \approx \beta_i^{(0)} (i = 1, 2, ..., n-1)$. Hence, from the second iteration on, it is more important to reduce the other $\beta_i (i = 1, 2, ..., n-1)$ instead of β_n. This may, in some sense, explain that a larger α_k (such as BB1) is better than a smaller α_k(such as BB2).

However, we would have a different picture when an iterate point x_k has the property that $|\beta_i^{(k-1)}| (i = 1, 2, ...n)$ are in the same order. For simplicity, we suppose that

$$|\beta_i^{(k-1)}|^2 \approx \|g_{k-1}\|_2^2/n, \tag{3.5}$$

for all $i = 1, 2, ..., n$. Thus, we would have

$$\alpha_k^{BB1} \approx \frac{n}{\sum_{j=1}^n \lambda_i}, \tag{3.6}$$

and

$$\alpha_k^{BB2} \approx \frac{\sum_{j=1}^n \lambda_i}{\sum_{j=1}^n \lambda_i^2}. \tag{3.7}$$

These give that

$$\alpha_k^{BB1} \approx \frac{n}{\lambda_n} >> \frac{1}{\lambda_n} \approx \alpha_k^{BB2}. \tag{3.8}$$

In this case, it can be easily seen that normally the shorter BB step $\alpha_k = \alpha_k^{BB2}$ would give a smaller $\|g_{k+1}\|$. Therefore, it is reasonable for us to believe the shorter BB step (BB2) would be efficient if we want to obtain a very accurate solution of a very large scale and ill-conditioned problem.

Hence, we would like to investigate the behavior of the BB2 step and shorter BB2 steps. What made us to explore the shorter steps is the curiosity on why BB2 in general performs worse than BB1. Another motivation is our belief that for very large scale and ill-conditioned problems a shorter step may be more efficient than a larger step. We consider the short BB2 step:

$$\alpha_k = \alpha_k^{SBB(m)} = \min_{\max[0,k-m]\leq j\leq k} \alpha_j^{BB2}, \tag{3.9}$$

where m is a given non-negative integer. If $m = 0$, the step-length (3.9) is nothing but the BB2 step. For $m > 0$, α_k given by (3.9) is not larger and may be smaller than α_k^{BB2}. Thus we call the method (1.5) with (3.9) the short BB method (SBB). The step-length defined by (3.9) satisfies (2.2), which means that the SBB method always converges for convex quadratic functions.

4 Numerical Results

In this section, we test our SBB method, namely the gradient method (1.5) with (3.9). We call our SBB method with parameter m by SBB(m). Different parameters $m = 1, 2, 3, 4, 9$ and 19 are used. We compared our algorithms with BB1, BB2 and the adaptive BB (ABB) of Zhou et. al.[22]. For the ABB method, $\kappa = 0.25$ is used.

The problem we used to compare the algorithms is the one suggested by Yuan[20]. The function to be minimized has the following form:

$$f(x) = (x - x^*)^T Diag(\lambda_1, \cdots, \lambda_n)(x - x^*). \quad x \in \Re^n. \tag{4.1}$$

The diagonal structure of the Hessian of the objective function does not lose generality because the gradient method is invariant with respect to orthogonal transformations. We test problems from small scale to large scale, with $n = 10^i (i = 1, 2, 3, 4, 5, 6)$. The solution vector $x_i^* (i = 1, ..., n) \in (-5, 5)$ are randomly generated. We let $\lambda_1 = 1$ and $\lambda_n = Cond(= 10^L, L = 1, 2, 3, 4, 5, 6)$ which is the condition number of the Hessian of function $f(x)$. $\lambda_i (i = 2, \cdots, n-1)$ are randomly chosen in the interval $(1, \lambda_n)$. For all problems the initial point is the zero vector $(0, \cdots, 0)^T$. We use two stop conditions. One is

$$\|g_k\|_2 \leq 10^{-5} \|g_0\|_2, \tag{4.2}$$

and the other is

$$\|g_k\|_2 \leq 10^{-5}. \tag{4.3}$$

The numerical results with the two different stopping conditions (4.2) and (4.3) are reported in Table 1 and Table 2 separately. For each case (different n and different λ_n), 10 runs are made and the average numbers of iterations required by each algorithm are listed. For each case, The least average iteration number is given in bold font to indicate the winner amongst all the algorithms.

n	λ_n	ABB	BB1	BB2	m=2	m=3	m=4	m=5	m=10	m=20
10	10	19.1	19.1	**18.1**	18.7	19.0	19.8	22.0	22.5	29.9
10	10^2	**42.7**	47.8	54.5	45.1	49.9	52.2	46.5	52.9	67.7
10	10^3	**51.9**	103.2	92.9	87.8	77.7	82.6	72.1	73.6	76.2
10	10^4	76.2	169.6	199.3	106.6	102.4	77.4	72.3	**59.0**	62.8
10	10^5	22.2	21.6	24.0	**20.9**	21.0	22.5	22.2	26.5	34.2
10	10^6	22.1	23.2	22.1	22.4	21.8	**21.6**	23.0	28.0	33.9
10^2	10	19.4	19.4	20.0	19.0	**18.9**	20.8	21.2	28.3	29.8
10^2	10^2	**46.6**	50.9	56.8	56.5	53.2	55.7	57.1	57.2	64.8
10^2	10^3	96.3	110.2	109.0	116.0	111.0	103.2	**94.3**	103.5	125.0
10^2	10^4	149.9	171.6	211.1	168.8	172.9	167.7	144.1	**130.3**	141.9
10^2	10^5	**75.8**	92.5	124.9	99.2	93.1	81.0	82.1	85.1	95.1
10^2	10^6	77.9	88.2	92.0	86.1	**74.4**	78.0	81.5	79.0	96.7
10^3	10	19.0	19.0	20.3	**18.7**	19.0	20.5	21.0	26.7	30.0
10^3	10^2	**50.1**	53.7	56.5	56.4	53.4	60.0	59.7	59.9	65.5
10^3	10^3	**103.3**	109.2	111.8	108.7	104.3	**103.3**	111.5	113.2	133.5
10^3	10^4	106.7	109.2	109.3	107.7	105.6	105.5	**105.0**	108.6	129.3
10^3	10^5	111.4	**108.1**	124.9	120.8	117.2	110.4	123.2	119.2	129.3
10^3	10^6	103.2	107.3	114.1	114.8	108.8	**100.6**	109.5	109.2	124.9
10^4	10	19.0	19.0	19.9	**18.8**	19.0	20.2	21.0	26.3	30.0
10^4	10^2	**52.6**	54.0	56.3	54.9	55.2	55.7	56.3	57.9	65.2
10^4	10^3	**101.4**	111.3	117.2	108.1	101.5	103.7	107.7	106.9	126.0
10^4	10^4	116.6	120.0	124.5	127.0	117.5	**115.8**	121.4	118.3	140.9
10^4	10^5	117.7	123.0	119.0	117.5	119.4	112.0	112.3	**111.8**	132.8
10^4	10^6	**107.6**	121.6	116.0	118.3	111.1	116.2	114.7	117.6	132.5
10^5	10	**19.0**	**19.0**	20.0	**19.0**	**19.0**	20.0	21.0	26.0	30.0
10^5	10^2	53.7	52.7	59.0	**52.2**	54.4	55.0	53.6	57.1	65.0
10^5	10^3	101.9	96.5	110.9	108.5	**95.9**	102.0	104.4	116.5	121.0
10^5	10^4	**111.6**	121.8	123.8	118.2	115.5	116.7	121.7	119.9	141.8
10^5	10^5	**110.9**	117.2	120.0	117.7	116.7	124.4	124.1	120.8	142.4
10^5	10^6	117.6	**115.5**	129.6	125.3	123.7	120.9	124.3	120.9	142.6
10^6	10	**19.0**	**19.0**	20.0	**19.0**	**19.0**	20.0	21.0	26.0	30.0
10^6	10^2	52.0	51.3	59.5	49.8	54.4	56.0	**48.1**	57.0	65.0
10^6	10^3	94.5	103.1	110.7	108.4	**94.0**	98.1	102.3	111.3	121.0
10^6	10^4	110.8	116.8	123.6	122.2	113.1	112.4	124.3	120.3	148.7
10^6	10^5	125.5	**114.4**	118.6	120.8	121.1	115.6	122.8	120.3	149.0
10^6	10^6	**113.6**	113.9	123.6	119.1	**113.6**	114.7	122.3	120.3	149.0

TABLE 1. Iteration numbers of different gradient methods ($\|g_k\|_2 \leq 10^{-5}\|g_0\|_2$)

When the stopping condition is (4.2), from Table 1 we find that the ABB method is the winner, as it wins 14 times out of the all 36 cases. Ranking by achieving the least number of iterations, the next best algorithms are SBB(2), SBB(1) and BB1, with winning on 7, 6 and 5 cases respectively. Table 1 also shows that BB1 is much better than BB2 as expected. If we make a one-to-one comparison between BB1 and BB2 on all the 36 cases, we find that BB2 wins only 5 cases while BB1 wins 31 cases. These results agree with the general belief that BB1 is better than BB2.

n	λ_n	ABB	BB1	BB2	m=1	m=2	m=3	m=4	m=9	m=19
10	10	24.4	24.4	25.3	**24.2**	24.9	25.3	26.3	32.7	32.7
10	10^2	66.7	81.4	85.6	69.5	72.9	73.3	69.4	**65.4**	76
10	10^3	98.6	200.3	228.1	183.6	127.7	125.1	105.7	92.4	**91.6**
10	10^4	117.1	679.5	608.0	286.3	158.5	116.7	103.6	**77.4**	85.4
10	10^5	329.8	2235.0	1148.0	356.5	169.7	122.9	105.2	**92.3**	95
10	10^6	1011.6	3052.3	908.4	307.7	152.2	137.1	100.3	**97.6**	102.2
10^2	10	30.3	30.3	30.4	30.6	30.7	30.0	**29.4**	34.1	35.6
10^2	10^2	**85.8**	98.5	106.5	98.3	96.5	93.8	93.4	101.8	111.2
10^2	10^3	197.4	328.8	318.2	300.4	277.5	258.3	223.2	**179**	182.8
10^2	10^4	310.4	989.9	937.1	879.9	640.7	502.5	437.1	251.5	**220.6**
10^2	10^5	774.5	2872.9	1903.9	1261.4	792.5	599.5	480.6	260.3	**225.6**
10^2	10^6	1214.4	4522.3	2038.0	1114.8	672.7	537.7	443.8	258.2	**236.4**
10^3	10	32.2	32.2	**30.5**	31.8	31.8	33	31.6	36.1	38.0
10^3	10^2	**101.1**	109.8	104.1	109.2	106.8	105.5	107.3	106.6	121.1
10^3	10^3	286.8	343.5	385.4	341.2	305.3	319.8	283.5	217.9	**216.1**
10^3	10^4	530.3	1140.3	1149.2	965.8	819.9	694.3	521.0	337.4	**302.3**
10^3	10^5	988.9	2927.7	2443.9	1638.9	1096.4	934.0	717.8	401.5	**346**
10^3	10^6	1587.5	4534.8	2310.5	1590.7	1076.2	858.2	693.9	384.6	**333.7**
10^4	10	34.3	34.3	**31.1**	32.6	33.1	37.7	34.1	37.0	48.2
10^4	10^2	**106.9**	115.0	113.3	111.2	108.8	114.2	113.6	114.0	133.3
10^4	10^3	334.4	380.7	370.2	385.1	351.8	321.7	304.9	**233.1**	233.5
10^4	10^4	921.6	1272.5	1250.8	1071.9	888.8	742.6	646.0	387.4	**338.4**
10^4	10^5	1610.6	3226.1	2574.3	1827.0	1424.7	1014.5	940.8	502.0	**425.1**
10^4	10^6	2316.0	5954.0	3404.7	1993.3	1602.7	1271.6	1069.7	534.6	**454.1**
10^5	10	35.0	35.0	**33.5**	35.7	35.3	39.3	38.4	38.0	57.0
10^5	10^2	**115.3**	119.3	121.7	117.5	115.6	117.7	122.2	120.5	150.3
10^5	10^3	349.5	402.0	416.8	392.2	367.9	325.9	317.9	**239.3**	244.8
10^5	10^4	1055.7	1341.9	1195.8	1057.7	869.4	748.5	645.7	389.3	**357.1**
10^5	10^5	2291.1	2640*	2475.3	2088.8	1527.6	1217.6	979.6	559.4	**461.0**
10^5	10^6	2140*	5120*	3149*	2445.1	1924.8	1571.7	1263.5	654.9	**531.5**
10^6	10	36.9	36.9	**35.2**	36.7	36.5	39.4	41.0	40.0	58.0
10^6	10^2	**121.1**	123.4	131.5	129.3	125.3	123.0	123.1	122.3	150.3
10^6	10^3	369*	376*	461*	470*	410*	387*	303*	243*	**220***
10^6	10^4	1154*	1130*	1325*	1128*	977*	760*	613*	406*	**364***
10^6	10^5	Fail	Fail	Fail	Fail	1557*	1192*	1023*	520*	**456***
10^6	10^6	Fail	Fail	Fail	Fail	Fail	1357*	1343*	652*	**567***

TABLE 2. Iteration numbers of different gradient methods ($\|g_k\|_2 \leq 10^{-5}$)

Now, let us discuss Table 2, where the results with the stopping condition (4.3) are given. Since in general the stopping condition (4.3) is more strict than (4.2), some algorithms fail to find a solution within the maximum allowed CPU time, which is set to 10 minutes. In Table 2, a number followed by a superscript "$*$" is the iteration number of a single run instead of the average of 10 runs. While "Fail" indicates that even a single run failed to find a solution within 10 minutes.

It is surprising to find that in Table 2 the winner now is SBB(19), which wins 18 cases out of all the 36 cases, particularly it wins all the cases when both n and λ_n are large. Please notice that the only change that makes Table 1 and Table 2 different is the stopping condition. For a given case, the 10 repeated randomly generated problems for both Table 1 and Table 2 are also the same. This shows that SBB(19) use far less numbers of iterations to reduce $\|g_k\|_2$ from $10^{-5}\|g_0\|_2$ to 10^{-5}. Another interesting point is that now BB2 performs much better than BB1, which is unexpected. If we have a one-to-one comparison between BB1 and BB2, we find that BB2 wins 23 cases while BB1 wins only 11 cases. Particularly, BB2 wins over BB1 for all the very ill-conditioned cases, namely when $\lambda_n = 10^5$ or $\lambda_n = 10^6$.

Now we consider two specific distributions of the eigenvalues $\lambda_i (i = 2, ..., n-1)$. The first case is

$$\frac{\lambda_{i+1}}{\lambda_i} = \left(\frac{\lambda_n}{\lambda_1}\right)^{\frac{1}{n-1}}, \quad i = 1, 2, ..., n-1. \tag{4.4}$$

And the second case is to have $\lambda_i (i = 2, ..., n-1)$ equally distributed in the interval $(1, \lambda_n)$. To be more exact, we let $\lambda_{i+1} - \lambda_i$ be constant:

$$\lambda_{i+1} - \lambda_i = \frac{\lambda_n - \lambda_1}{n-1}, \quad i = 1, 2, ..., n-1. \tag{4.5}$$

In both cases, the stopping condition is

$$\|g_k\|_2 \leq 10^{-6}\|g_0\|_2. \tag{4.6}$$

The solution is chosen by vector $x_i^* (i = 1, ..., n) \in (-0.5, 0.5)$ randomly, and the average number of iterations out of 10 runs with the fixed starting point $(0, 0, ..., 0)^T$ are given in Table 3 and Table 4 for the two cases respectively.

n	λ_n	ABB	BB1	BB2	m=1	m=2	m=3	m=4	m=9	m=19
10	10	25.3	25.3	25.5	**24.7**	25.3	25.5	26.2	34.3	32.4
10^3	10	25.4	25.4	26.3	25.7	25.9	**23.1**	24.3	32.0	32.2
10^5	10	25.0	25.0	26.0	25.0	26.0	**23.0**	25.0	32.0	33.0
10	10^3	178.9	173.5	169.1	149.1	113.5	103.7	**95.5**	101.6	132.9
10^3	10^3	200.3	245.4	246.8	230.4	230.8	211.8	212.2	187.5	**179.7**
10^5	10^3	196.5	215.8	240.0	236.8	202.5	202.4	213.0	**193.7**	222.0
10	10^5	836.9	802.8	720.7	203.6	130.0	**118.7**	127.2	136.7	179.5
10^3	10^5	1262.6	1444.1	1702.1	1391.2	1148.5	966.6	832.8	477.9	**377.6**
10^5	10^5	1297.5	1482.5	1447.0	1406.7	1291.2	1175.5	1025.7	678.8	**437.3**

TABLE 3. Iteration numbers of different gradient methods when λ_{i+1}/λ_i is constant.

From Table 3, the numerical results favor our SBB method over the ABB, BB1 and BB2 methods. Particularly, for very ill-conditioned problems, namely problems with $\lambda_n = 10^5$, our SBB method performs much better than ABB, BB1 and BB2. Therefore, we believe that for ill-conditioned problems with $\lambda_{i+1}/\lambda_1 \approx$ *constant*, the SBB method will be much faster than ABB, BB1 and BB2.

n	λ_n	ABB	BB1	BB2	m=1	m=2	m=3	m=4	m=9	m=19
10	10	22.0	22.0	23.9	**21.6**	23.0	24.0	25.0	33.8	32.5
10^3	10	24.4	24.4	25.7	23.5	23.4	**22.9**	24.1	33.0	32.0
10^5	10	24.0	24.0	26.0	**23.0**	**23.0**	**23.0**	24.0	33.0	32.0
10	10^3	**54.8**	130.4	131.7	125.8	96.1	108.4	107.8	96.5	70.5
10^3	10^3	**145.6**	163.7	168.4	157.8	162.3	160.7	155.1	147.1	177.0
10^5	10^3	**144.5**	158.0	163.0	155.9	146.5	152.5	153.0	149.7	168.5
10	10^5	237.9	382.1	549.9	280.3	158.3	111.6	104.5	**77.0**	109.7
10^3	10^5	151.4	164.0	185.7	164.4	165.0	155.0	165.4	**148.8**	159.2
10^5	10^5	262.0	286.4	295.3	274.7	266.9	282.7	268.1	**238.5**	244.1

TABLE 4. Iteration numbers of different gradient methods when $\lambda_{i+1} - \lambda_i$ is constant.

From Table 4, we can see that all the algorithms perform more or less the same. Actually, the ABB method, which wins when $\lambda_n = 10^3$ for $n = 10$, 10^3 and 10^5, can be regarded as the overall best method when the stopping condition is (4.6). Similar to the phenomenon revealed in Tables 1 and 2, we also observe that our SBB method will outperform ABB method if a more accurate solution is needed. For example, let us consider the situation when $n = 10^3$ and $\lambda_n = 10^3$, which is the case that ABB wins for the stopping condition (4.6). If the stopping condition is replaced by $\|g_k\|_2 \le 10^{-9}\|g_0\|_2$ the ABB method needs 263.6 iterations while the SBB(9) method needs 243.7 iterations. If we use an even more strict stopping condition $\|g_k\|_2 \le 10^{-12}\|g_0\|_2$, the ABB method would need 380.2 iterations against 304.3 iterations by the SBB(9) method. Even for the situation when $n = 10$ and $\lambda_n = 10^3$, for which the ABB method preforms much better than the other methods under the stopping condition (4.6), we find that the ABB method requires 126.0 iterations comparing 117.2 iterations by SBB(9) method if the stopping condition is replaced by $\|g_k\|_2 \le 10^{-13}\|g_0\|_2$. Of course, in real applications it is unlikely to require such high accurate solutions.

For many practical problems, matrix A is obtained by finite difference approximation to Laplace's equation[9, 22]. For such A, we can easily see that the difference $\lambda_{i+1} - \lambda_i$ are of the same magnitude for many i. Therefore, we expect that for such problems derived from Laplace equations the best gradient method to use is the ABB method. Indeed, we tested the Laplace1(b) problem of Fletcher[9] in which A is defined by

$$A = \begin{bmatrix} W & -I & & & \\ -I & W & -I & & \\ & -I & W & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & W \end{bmatrix} \in \Re^{10^6 \times 10^6} \tag{4.7}$$

where

$$W = \begin{bmatrix} T & -I & & & \\ -I & T & -I & & \\ & -I & T & \ddots & \\ & & \ddots & \ddots & -I \\ & & & -I & T \end{bmatrix} \in \Re^{10^4 \times 10^4}, \qquad T = \begin{bmatrix} 6 & -1 & & & \\ -1 & 6 & -1 & & \\ & -1 & 6 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 6 \end{bmatrix} \in \Re^{10^2 \times 10^2}. \tag{4.8}$$

It is known[9] for this matrix A we have $\lambda_n/\lambda_1 \approx 4133.6$. In Table 5, we give the numbers of iterations needed for all the algorithms with different stopping conditions

$$\|Ax_k - b\|_2 \le \theta\|b\|_2, \qquad \theta = 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}, 10^{-9}. \tag{4.9}$$

The starting point is $x_0 = (0, 0, ..., 0)^T$ for all the runs.

θ	ABB	BB1	BB2	m=1	m=2	m=3	m=4	m=9	m=19
10^{-4}	173	176	157	178	200	199	**166**	181	225
10^{-5}	**276**	394	392	278	289	298	290	322	417
10^{-6}	387	462	611	374	426	558	**361**	442	605
10^{-7}	460	510	864	478	**458**	760	493	652	701
10^{-8}	**570**	590	1017	737	601	844	645	820	759
10^{-9}	**590**	611	1062	775	819	851	676	881	942

TABLE 5. Iterations for Laplace1(b) with different stopping conditions

Our results in Table 5 confirm the finding of Zhou et. al.[22] that the ABB method is better than the BB1 method. Moreover, for this specific problem, BB1 is much better than BB2. Though our short BB2 steps do provide improvements over the original BB2 method, the SBB methods perform not yet as good as the ABB method for this Laplace1(b) problem. Therefore it is reasonable for us to believe that, if we want to find a better gradient method than the ABB method for such problems derived from Laplace equations, we might need to explore special step-lengths which make good use of the special eigenvalue distributions of such matrices. This is an interesting and important problem to study because many practical problems are derived from Laplace equations.

5 Discussion

In this chapter we have generalized the convergence result for the gradient method with retards by Friedlander et. al.[10] from (2.1) to (2.2). Our simple generalization allows more choices of step-lengths α_k. We give an intuitive analysis on the impact of the step-length of the gradient method for large scale and ill-conditioned problems, and believe that short step-lengths in the interval (2.2) should perform better than long step-lengths. We propose the short BB2 (SBB) method, which uses the smallest value of all the BB2 step-lengths in the previous m iterations as the step-length. Numerical results on large scale and ill-conditioned problems show that the SBB method performs better than the BB methods and the adaptive BB method if a high accurate solution is needed. Our numerical results also reveal that BB2 is better than BB1 when we need to find a very high accurate solution for large scale and ill-conditioned problems. This is, to some extent, a surprising discovery because in general it has been widely regarded that BB1 is better than BB2. Our numerical results also suggest that corresponding special step-lengths might be needed to construct efficient gradient methods for solving problems with certain special eigenvalue distributions such as those problems derived from Laplace equations.

References

[1] J. Barzilai and J. M. Borwein, *Two point step size gradient methods*, IMA J. Numer. Anal., 8(1988) 141-148.

[2] Y.H. Dai, *Alternate step gradient method*, Optimization 52(2003) 395-415.

[3] Y.H. Dai and R. Fletcher, *On the asymptotic behaviour of some new gradient methods*, Math. Program. 13(2005) 541-559.

[4] Y,H. Dai and L.Z. Liao, *R-linear convergence of the Barzilai and Borwein gradient method*, IMA J. Numer. Anal. 22(2002) 1-10.

[5] Y.H. Dai, J.Y. Yuan, and Y. Yuan, *Modified two-point step-size gradient methods for unconstrained optimization*, Computational Optimization and Applications, 22(2002), 103-109.

[6] Y.H. Dai and Y. Yuan, *Alternate minimization gradient method*, IMA Journal of Numerical Analysis, 23(2003), 377-393.

[7] Y.H. Dai and Y. Yuan, *Analysis of monotone gradient methods*, J. Industrial and Management Optimization, 1(2005) 181-192.

[8] Y. H. Dai and H. Zhang, *An adaptive two-Point step-size gradient method*, Numerical Algorithm, 27(2001) 377-385.

[9] R. Fletcher, *On the Barzilar-Borwein method*, Research Report, University of Dundee, UK, 2001.

[10] A. Friedlander, J. M. Martínez, B. Molina, and M. Raydan, *Gradient method with retards and generalizations*, SIAM J. Numer. Anal., 36(1999), 275-289.

[11] J. Nocedal, A. Sartenaer and C. Zhu, *On the behavior of the gradient norm in the steepest descent method*, Computational Optimization and Applications 22(2002) 5-35.

[12] M. Raydan, *On the Barzilai and Borwein choice of steplength for the gradient method*, IMA J. Numer. Anal. 13(1993) 321-326.

[13] M. Raydan, *The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*, SIAM J. Optim., 7(1997) 26-33.

[14] M. Raydan and B. F. Svaiter, *Relaxed Steepest Descent and Cauchy-Barzilai-Borwein Method*, Computational Optimization and Applications, 21(2002) 155-167.

[15] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas, *A class of gradient unconstrained minimization algorithms with adaptive step-size*, J. Comp. and Appl. Math. 114(2000) 367-386.

[16] Y.F. Wang, Y. Yuan and H.C. Zhang, *A trust region-CG algorithm for delurring problem in atmospheric image reconstruction*, Science in China 45(2002) 731-740.

[17] Y.F. Wang and Y. Yuan, *On the regularity of a trust region-CG algorithm for nonlinear ill-posed inverse problems*, in: T. Sunada, P.W. Sy and L. Yang, Eds., Proceedings of the Third Asian Mathematical Conference (World Scientific, Singapore, 2002) pp.562-580.

[18] Y.F. Wang and Y. Yuan, *A trust region method for solving distributed parameter identification problems*, Journal of Comp. Math. 21(2003) 759-772.

[19] Y.F. Wang and Y. Yuan, *Convergence and regularity of trust region methods for nonlinear ill-posed inverse problems*, Inverse Problems, 21(2005) 821-838.

[20] Y. Yuan, *A new stepsize for the steepest descent method*, Journal of Comp. Math. 24(2006) 149-156.

[21] Y. Yuan, *Step-sizes for the gradient method*, in: K.S. Liu, Z.P. Xin and S.T. Yau, eds., Third International Congress of Chinese Mathematicians (AMS/IP Studies in Advanced Mathematics, 2008), pp. 785-796.

[22] B. Zhou, L. Gao and Y.H. Dai, *Gradient methods with adaptive stp-sizes*, Computational Optimization and Applications, 35(2006) 69-86.