Analyses of Monotone Gradient Methods

Yu-Hong Dai[1,2] and Ya-xiang Yuan[1]

Dedicated to Professor Ji-ye Han on the occasion of his 70 birthday

Abstract. The gradient method is one simple method in nonlinear optimization. In this paper, we give a brief review on monotone gradient methods and study their numerical properties by introducing a new technique of long-term observation. We find that, one monotone gradient algorithm which is proposed by Yuan recently shares with the Barzilai-Borwein (BB) method the property that the gradient components with respect to the eigenvectors of the function Hessian are decreasing together. This might partly explain why this algorithm by Yuan is comparable to the BB method in practice. Some examples are also provided showing that the alternate minimization algorithm and the other algorithm by Yuan may fall into cycles. Some more efficient gradient algorithms are provided. Particularly, one of them is monotone and performs better than the BB method in the quadratic case.

Key words: gradient method, monotone, nonmonotone, strictly convex quadratics, cycle.

AMS(MOS) subject classifications. 65k, 90c.

1. Introduction

To minimize a smooth function f, we are interested in the gradient method

$$x_{k+1} = x_k - \alpha_k g_k, \tag{1.1}$$

where $g_k = g(x_k) = \nabla f(x_k)$ is the gradient of $f(x)$ at the current iteration x_k and α_k is a stepsize. The steepest descent (SD) method, which can be dated back to Cauchy [3], calculates the stepsize by an exact line search

$$\alpha_k^{SD} = \arg\min_{\alpha \in \Re} f(x_k - \alpha g_k). \tag{1.2}$$

[1] State Key Laboratory of Scientific and Engineering Computing, Institute of Computational Mathematics and Scientific/Engineering computing, Academy of Mathematics and System Sciences, Chinese Academy of Sciences, P. O. Box 2719, Beijing 100080, China. Email: {dyh,yyx}@lsec.cc.ac.cn

[2] Department of Mathematics, University of Bayreuth, D-95440, Germany.

It is well known that the SD method is very slow and produces the so-called zigzagging phenomenon. Assume that the objective function

$$f(x) = \frac{1}{2}x^T Ax - b^T x, \tag{1.3}$$

where $A \in \Re^{n\times n}$ is symmetric and positive definite and $b \in \Re^n$ is some vector. Akaike [1] proved that the SD method is Q-linearly convergent with the Q-linear factor $\frac{\kappa-1}{\kappa+1}$, where κ is the condition number of the Hessian matrix A. Forsythe [9] gives an interesting analysis to show that the gradients $g(x_k)$ will tend to zero eventually along two directions alternatively.

Another ingenious way of choosing α_k is provided by Barzilai and Borwein [2]. Their basic idea is to regard the matrix $D_k = \frac{1}{\alpha_k}I$ as some approximation to the Hessian $\nabla^2 f(x_k)$ and then impose some quasi-Newton property on D_k. Mathematically, defining $s_{k-1} = x_k - x_{k-1}$ and $y_{k-1} = g_k - g_{k-1}$, they decide the stepsize for $k \geq 2$ by

$$\alpha_k = \arg\min_{\alpha\in\Re} \|D_k s_{k-1} - y_{k-1}\|_2, \tag{1.4}$$

which yields

$$\alpha_k^{BB} = \frac{s_{k-1}^T s_{k-1}}{s_{k-1}^T y_{k-1}}. \tag{1.5}$$

Barzilai and Borwein proves that the method (1.1)–(1.5) is R-superlinearly convergent for 2-dimensional convex quadratics. In practice, the choice (1.5) has been shown to be much more efficient than (1.2) (for example, see Fletcher [8]). The BB-like method has now received much attention in the optimization community (see [6, 4] and the references therein).

Although the BB method greatly speeds up the convergence, it cannot guarantee a descent on the objective function at every iteration. Therefore one interesting question is, does there exist some stepsize formula which enables fast convergence and possesses the monotone property? Presently, there have been several studies on this topic. Specifically, Yuan [11] provides a very interesting formula for the stepsize α_k and proposes two algorithms, Algorithm 2.1(A) and Algorithm 2.1(B), based on the new stepsize (see (2.10)). The numerical experiments in [11] show that Algorithm 2.1(B) is comparable to the BB method for large scale problems and better for small scale problems. In this paper, we will give a brief review on monotone gradient methods (see §2) and provide some analyses for these methods (see §3 and §4). These analyses explain why Algorithm 2.1(B) is better than other existing monotone gradient methods and comparable to the BB method. Some new efficient choices based on (2.10) or its variants are given in §5. Conclusion and discussion are made in the last section. Particularly, one of them is monotone and performs better than the BB method.

Throughout this paper, we assume that the objective function has the form (1.3).

2. A brief review on monotone gradient methods

If the method is such that $f(x_k) < f(x_{k+1})$ for all $k \geq 1$, then we call the method is a monotone method. The SD method is clearly a monotone method due to the choice (1.2). Under the assumption that f is given in (1.3), we have for any gradient method (1.1) that

$$f(x_{k+1}) = f(x_k) - \alpha_k g_k^T g_k + \frac{1}{2}\alpha_k^2 g_k^T A g_k. \tag{2.1}$$

By (1.2) and the above relation, we can obtain the analytical expression of the SD stepsize

$$\alpha_k^{SD} = \frac{g_k^T g_k}{g_k^T A g_k}. \tag{2.2}$$

Further, by writing (2.1) as

$$f(x_{k+1}) = f(x_k) + \frac{1}{2}\alpha_k(\alpha_k - 2\alpha_k^{SD}) g_k^T A g_k,$$

we know that in the quadratic case a gradient method is monotone if and only if

$$0 < \alpha_k < 2\alpha_k^{SD}. \tag{2.3}$$

Assume that λ_1 and λ_n are the minimal and maximal eigenvalues of the Hessian matrix A, respectively. The following choice for the stepsize is studied in Elman and Golub [7]

$$\alpha_k^{OPT1} = \frac{2}{\lambda_1 + \lambda_n} \tag{2.4}$$

This stepsize is such that the modulus $\|I - \alpha A\|_2$ is minimized, that is

$$\alpha_k^{OPT1} = \arg\min_{\alpha\in\Re^1} \|I - \alpha A\|_2,$$

and hence gives the best convergence result while the analysis is via the relation

$$\|g_{k+1}\|_2 \le \|I - \alpha_k A\|_2 \|g_k\|_2.$$

Since $g_k^T A g_k \le \lambda_1 g_k^T g_k$, the choice (2.4) clearly satisfies (2.3). However, since λ_1 and λ_n is normally unknown to users, the method (1.2)–(2.4) is not practical unless λ_1 and λ_n are known or can be estimated beforehand. Dai and Yang [5] proposed the following stepsize formula

$$\alpha_k^{OPT2} = \frac{\|g_k\|_2}{\|Ag_k\|_2}. \tag{2.5}$$

It is obvious that $\alpha_k^{OPT2} \le \alpha_k^{SD}$. The choice (2.5) is shown to tend to α_k^{OPT1}. Meanwhile, the minimal eigenvalue λ_1 and the maximal eigenvalue λ_n of A can be estimated through the gradients generated. However, the numerical performance of (2.5) is similar to those of (1.2) and (2.4). All of them have the drawback that the gradients $g(x_k)$ tend to zero eventually along two directions alternatively.

In [6], we proposed the so-called alternate minimization (AM) gradient method, that carries out exact line searches in odd iterations and minimizes the gradient norm in even iterations. Namely,

$$\alpha_k^{AM} = \begin{cases} \frac{g_k^T g_k}{g_k^T A g_k}, & \text{if } k \text{ is odd;} \\ \frac{g_k^T A g_k}{g_k^T A^2 g_k}, & \text{if } k \text{ is even.} \end{cases} \tag{2.6}$$

The AM method is proved to be Q-superlinearly convergent for two-dimensional quadratics, and Q-linearly convergent in the any-dimension case. The numerical results in [6] show that that the AM method is significantly better than the SD method. To generalize the AM method for unconstrained optimization, we observed that a suitable reduction on the k-th SD step can lead to a better function value $f(x_{k+2})$. Following this idea, we propose two shortened SD step methods

$$\alpha_k^{SS1} = \gamma_1 \alpha_k^{SD} \tag{2.7}$$

and

$$\alpha_k^{SS2} = \begin{cases} \gamma_2 \alpha_k^{SD}, & \text{if } k \text{ is odd;} \\ \alpha_k^{SD}, & \text{if } k \text{ is even,} \end{cases} \tag{2.8}$$

where γ_1 and γ_2 are some positive constants less than 1. It is suggested in [6] that $\gamma_1 = 0.8$ and $\gamma_2 = 0.75$. Though the modifications are simple, the SS1 and SS2 methods avoid the zigzagging phenomenon to much extent and is comparable to the AM method. In [10], Raydan and Svaiter gave the random choice

$$\alpha_k^{RAND} = \theta_k \alpha_k^{SD}, \tag{2.9}$$

where θ_k is randomly chosen with a uniform distribution on $[0, 2]$. This random SD method also much outperforms the SD method. However, despite the numerical improvements over the classical SD method, it is difficult for the AM, SS1, SS2, and random SD methods to beat the BB method when the problem condition is large.

Recently, Yuan [11] proposed an interesting choice for the stepsize

$$\alpha_k^{Y} = \frac{2}{\sqrt{(1/\alpha_{k-1}^{SD} - 1/\alpha_k^{SD})^2 + 4\|g_k\|_2^2/\|s_{k-1}\|_2^2} + 1/\alpha_{k-1}^{SD} + 1/\alpha_k^{SD}}. \tag{2.10}$$

An important property of this stepsize is that, for any 2-dimensional convex quadratic function, if $\alpha_1 = \alpha_1^{SD}$, $\alpha_2 = \alpha_2^{Y}$ and $\alpha_3 = \alpha_3^{SD}$, then x_4 gives the minimizer in exact arithmetic. Then based on this choice, Yuan proposed two gradient algorithms, Algorithm 2.1(A) and Algorithm 2.1(B). They are corresponding to the stepsize formulae

$$\alpha_k^{YA} = \begin{cases} \alpha_k^{SD}, & \text{if } k \text{ is odd;} \\ \alpha_k^{Y}, & \text{if } k \text{ is even} \end{cases} \tag{2.11}$$

and

$$\alpha_k^{YB} = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,3) \neq 0; \\ \alpha_k^{Y}, & \text{if } mod(k,3) = 0. \end{cases} \tag{2.12}$$

Both of the Algorithms are monotone since it is easy to see from (2.10) that $\alpha_k^Y < 2\alpha_k^{SD}$. The numerical experiments in [11] show that Algorithm 2.1(B) is comparable to the BB method for large scale problems and better for small scale problems. However, it is also found that Algorithm 2.1(A) is far more worse than Algorithm 2.1(B).

3. Observing monotonic gradient methods

As the first author discussed with Roger Fletcher (private communications), the BB method has an important property in numerical computations, that is, the gradient components with respect to the eigenvectors of the function Hessian are decreasing together. In this section, we will provide a numerical analysis showing that Algroithm 2.1(B) also has this property, while the other monotone methods mentioned in the previous section do not have.

In the quadratic case, it follows from (1.2) and (1.3) that

$$g_{k+1} = (I - \alpha_k A) g_k. \tag{3.1}$$

Since any gradient method is invariant under any orthogonal transformations and the gradient components corresponding to the identical eigenvalues can be combined (for example, see Fletcher [8]), we assume that the matrix A is

$$A = diag(\lambda_1, \lambda_2, \ldots, \lambda_n) \quad \text{with } 0 < \lambda_1 < \lambda_2 < \cdots < \lambda_n. \tag{3.2}$$

Then denoting $g_k = (g_k^{(1)}, \ldots, g_k^{(n)})^T$, we have by (3.1) and (3.2) that

$$g_{k+1}^{(i)} = (1 - \alpha_k \lambda_i) g_k^{(i)} \qquad i = 1, 2, \ldots, n. \tag{3.3}$$

We assume that $g_k^{(i)} \neq 0$ for all k sufficiently large, since if $g_k^{(i)} = 0$, it follows from (3.3) that the component remains zero on all subsequent iterations, and hence can be disregarded. To observe how the gradient components with respect to the eigenvectors decrease, we define the following quantity

$$\xi_k = \frac{\min\{\sum_{j=0}^{L-1} |g_{k+j}^{(i)}|;\ i = 1, \ldots, n\}}{\max\{\sum_{j=0}^{L-1} |g_{k+j}^{(i)}|;\ i = 1, \ldots, n\}}, \tag{3.4}$$

where $L \geq 1$ is some integer. The basic role of L is to smooth the curve of ξ_k with $L = 1$. It is set to 100 in our tests.

Now let us consider one concrete example. We assume that the function is (1.3) and (3.2) where λ_i and g_1 are given by

$$\lambda_i = 11i - 10, \quad g_1^{(i)} = \sqrt{1+i}, \qquad \text{for } i = 1, \ldots, 10. \tag{3.5}$$

The dimension n is 10. This example is so easy that Algorithm 2.1(B) *etc.* only requires a few iterations to provide a satisfactory solution. Therefore to enable long-term observations, we normalize each gradient g_k before computing g_{k+1} by setting $g_k = g_k/\|g_k\|_2$, which does not affect the sequence $\{\xi_k\}$. In other words, we calculate the following sequences $\{\mu_k\}$ and $\{\nu_k\}$ instead of (3.1):

$$\left.\begin{array}{l}\nu_k := \nu_k \cdot \|\mu_k\|_2, \\ \mu_k := \mu_k/\|\mu_k\|_2, \\ \mu_{k+1} := (I - \alpha_k A)\mu_k\end{array}\right\} \quad \text{for } k = 1, 2, \ldots \tag{3.6}$$

where $\mu_1 = g_1$ and $\nu_1 = 1$. The calculations of α_k and ξ_k can be done with the sequence $\{\mu_k\}$. The above technique is simpler than the one used in [4].

With this technique, we plot the sequence $\{\xi_k\}$ generated by different gradient methods in Figure 3.1, where RAND, YA and YB stand for the random SD method (2.9), Algorithm 2.1(A) and Algorithm 2.1(B), respectively. All the computations are done with MATLAB (version 6.0.0.88). For the SS1 and SS2 methods, we use the suggested values $\gamma_1 = 0.8$ and $\gamma_2 = 0.75$ in [6].

From Figure 3.1, we see that the sequence $\{\xi_k\}$ tends to zero for all the tested monotone gradient methods except Algorithm 2.1(B). By the definition of ξ_k, we know that in this case, some component of the gradient approaches zero significantly faster than other gradient components. However, for both Algorithm 2.1(B) and the BB method, the sequence $\{\xi_k\}$ keeps relatively large and does not affect as the iteration number increases. We also tested other different examples and found that the above 10-dimensional is typical. To sum up, this numerical analysis shows that like the BB method, Algorithm 2.1(B) has the property that the gradient components with respect to the eigenvectors are decreasing together (In short, we call this property as "*descreasing together*"). This might explain why Algorithm 2.1(B) performs much better than Algorithm 2.1(A) and similarly to the BB method.

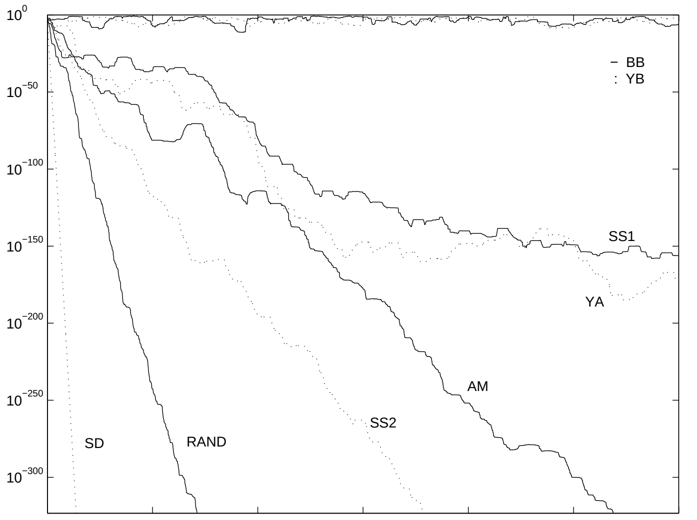

Figure 3.1: The sequence $\{\xi_k\}$ generated by different gradient methods

Further, if we define the set

$$\mathcal{B} = \{i : 1 \le i \le n,\ \liminf_{k\to\infty} |g_k^{(i)}|/\|g_k\|_2 \ne 0\} \tag{3.7}$$

and let n_b be the size of $\mathcal{B}$, namely, $n_b = |\mathcal{B}|$. Then for the SD method, the method (2.4), and the method (2.5), we know from the theroy in [9] and [5] that $n_b = 2$. Still taking the previous example, we have obtained Table 3.2.

Method	SD	RAND	SS1	SS2	AM	YA	YB	BB
n_b	2	2	3	2	4	7	10	10

Table 3.2: The value n_b for different gradient methods in the example

We have done a lot of experiments in which it always holds that $n_b^{SD} = n_b^{SS2} = n_b^{RAND} = 2$, $n_b^{YB} = n_b^{BB} = n$, and $n_b < n$ for all the monotone gradient methods considered in this paper except YB. This observation might be of helpful in estalishing some theoretical evidences for SS2 and RAND to explain why they are not so good as YB and BB. However, for SS1, SS2, AM and YA, we also observed that the value of n_b may be different from those in Table 3.2. This indicates that the theoretical analyses with these methods may be difficult.

4. Examples for the AM method and Algorithm 2.1(A)

The AM method and Algorithm 2.1(A) can be regarded the the main competitors of Algorithm 2.1(B). In this section, we provide several examples for the AM method and Algorithm 2.1(A). These examples expose the possible drawback of the methods, and demonstrate to some extent why Algorithm 2.1(B) is more efficient.

At first, we consider the following 3-dimensional example for the AM method

$$A = diag(1,\ 8,\ 15), \quad g_1 = (0.1,\ 0.2,\ 0.5)^T.$$

There is no necessity to mention the vector b in (1.3) since we only concern the gradient sequence $\{g_k\}$. We still use the technique appeared in (3.6) to observe the AM method. Instead of $\{\xi_k\}$, we observe the sequence of inverse stepsizes $\{\alpha_k^{-1}\}$. Figure 4.1 plots the subsequences $\{\alpha_{4i+j}^{-1}\}$ for $j = 1, 2, 3, 4$. From Figure 4.1, we see that each subsequence $\{\alpha_{4i+j}^{-1}\}$ and hence the whole sequence $\{\alpha_k^{-1}\}$ cycles. The period of the cycle with $\{\alpha_k^{-1}\}$ is $4 \times 17 = 68$. Further, we see that each subsequence $\{\alpha_{4i+j}^{-1}\}$ stays inside some interval that excludes the eigenvalues of A. Our numerical experiments show that

$$\{\alpha_{2i+1}^{-1}\} \subset [11.80,\ 13.95], \quad \{\alpha_{2i+2}^{-1}\} \subset [1.20,\ 7.20] \tag{4.1}$$

for all $i \ge 0$. The distance between the whole sequence $\{\alpha_k^{-1}\}$ and the maximal eigenvalue, 15, of the matrix A exceeds 1. Therefore by the relation (3.3), the AM method is only linearly convergent for 3-dimensional convex quadratics although

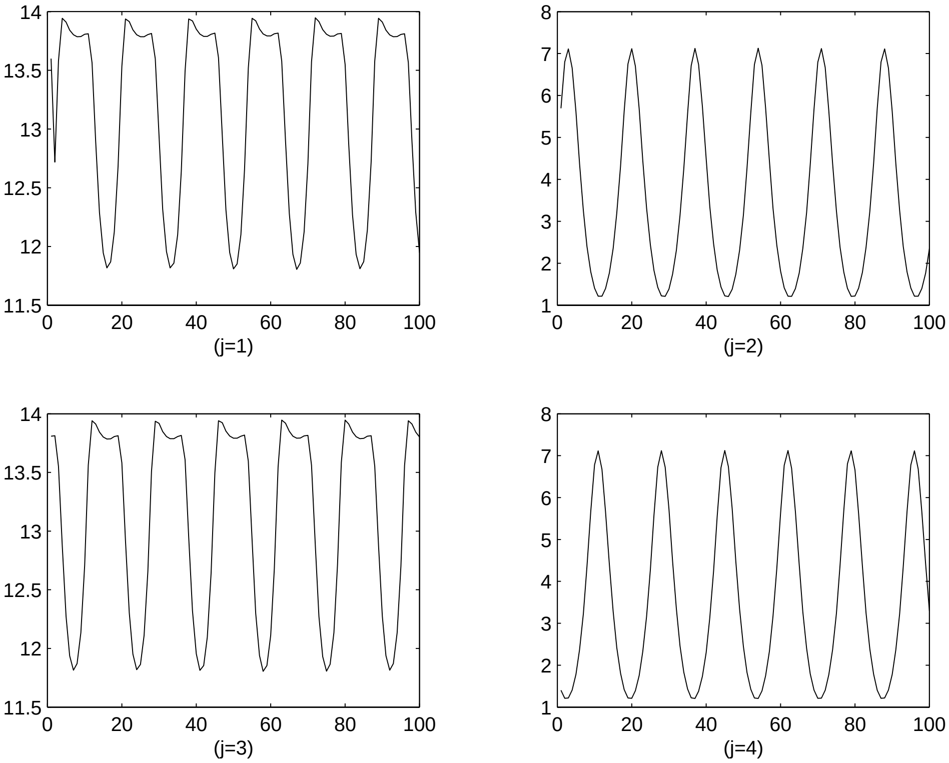

Figure 4.1: The subsequences $\{\alpha_{4i+j}^{-1}\}$; $(j = 1, 2, 3, 4)$ by AM for a 3-d example

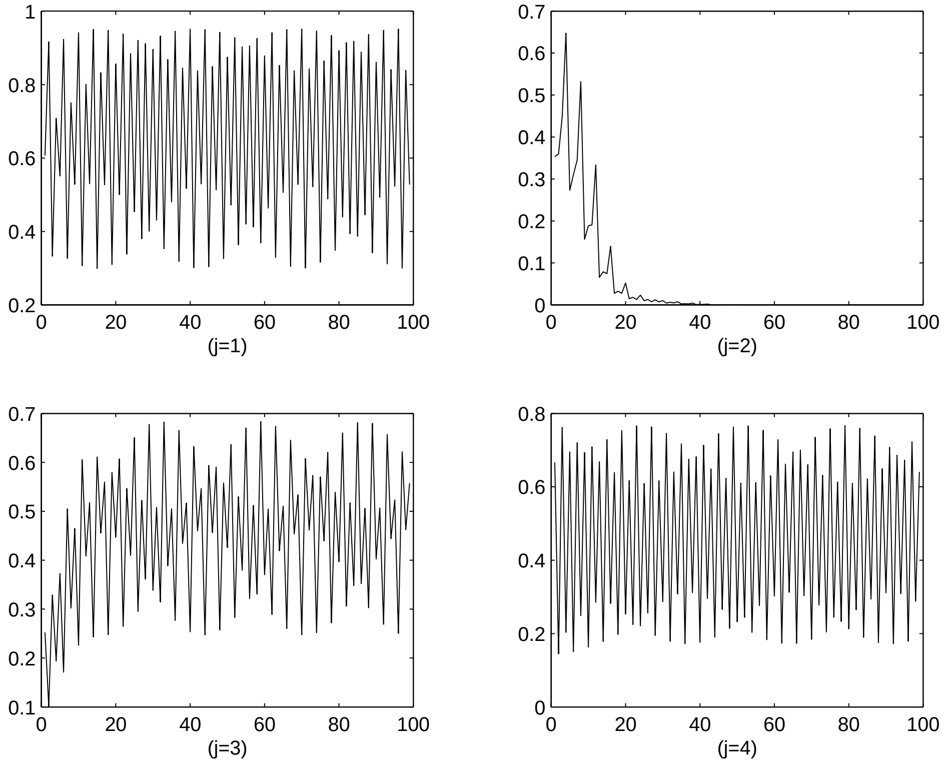

Figure 4.2: The subsequences $\{\frac{g_k^{(j)}}{\|g_k\|_2}\}$; $(j = 1, 2, 3, 4)$ by AM for a 4-d example

there is Q-superlinear convergence in the 2-dimensional case ([6]). From (4.1), we can also see that for this example, the AM method has *large stepsize–small stepsize* property, namely, a small stepsize will appear after each large stepsize and vice versa.

Next, we investigate a 4-dimensional example, where

$$A = diag(1,\ 5,\ 10,\ 15), \quad g_1 = (0.1,\ 0.2,\ 0.5,\ 1)^T.$$

Again, we observed that the sequence $\{\alpha_k^{-1}\}$ produces some kind of cycle and the sequence $\{\alpha_k^{-1}\}$ generated by the AM method has the "large stepsize–small stepsize" property. Specifically, we have that

$$\{\alpha_{2i+1}^{-1}\} \subset [12.50,\ 13.70], \quad \{\alpha_{2i+2}^{-1}\} \subset [1.90,\ 4.20] \tag{4.2}$$

for all $i \geq 1$. In this example, the second gradient component $g_k^{(2)}$ tends to zero faster than the others. See Figure 4.2 for the traces of different gradient components.

Now we turn to Algorithm 2.1(A) and consider the following 4-dimensional example where

$$A = diag(1,\ 2,\ 7,\ 8), \quad g_1 = (0.1,\ 0.2,\ 0.5,\ 1)^T.$$

j	$\frac{g^{(1)}_{\bar{k}+j}}{\|g_{\bar{k}+j}\|_2}$	$\frac{g^{(2)}_{\bar{k}+j}}{\|g_{\bar{k}+j}\|_2}$	$\frac{g^{(3)}_{\bar{k}+j}}{\|g_{\bar{k}+j}\|_2}$	$\frac{g^{(4)}_{\bar{k}+j}}{\|g_{\bar{k}+j}\|_2}$	$\alpha^{-1}_{\bar{k}+j}$
1	7.27610290e-01	6.83239238e-01	-5.13010519e-02	3.36988299e-02	1.49055592
2	5.81661104e-01	-5.67220588e-01	4.60592347e-01	-3.57471473e-01	3.66248783
3	5.45668293e-01	-3.32262650e-01	-5.41639889e-01	5.46327626e-01	4.95995821
4	6.96829452e-01	-3.17156985e-01	3.56333686e-01	-5.35599165e-01	7.56364527
5	9.31137060e-01	-3.59232587e-01	4.08888373e-02	4.75796878e-02	1.15492622
6	2.74509134e-01	5.77681706e-01	-4.54792213e-01	-6.19752067e-01	5.32953302
7	4.09380828e-01	6.62523954e-01	2.61686043e-01	5.70078734e-01	4.12474384
8	4.24879469e-01	4.67554228e-01	-2.49908442e-01	-7.33768527e-01	7.81717848

Table 4.3: The cycle of Algorithm 2.1(A) for a 4-d example

It is found that the stepsize $\{\alpha_k\}$ eventually tends to some cycle of eight different values. Table 4.3 listed the approximate values of these stepsizes and $\{\frac{g_k^{(i)}}{\|g_k\|_2}\}$ $(i = 1, 2, 3, 4)$. In the Table, $\bar{k}$ is some iteration number.

The above example also shows that Algorithm 2.1(A) is only linearly convergent for 4-dimensional convex quadratics. With the same Hessian matrix, we also tried some other initial values of g_1 and found that Algorithm 2.1(A) falls into the same cycle eventually. Similar cycles occur for some other matrices, for example $A = diag([1,\ 2,\ 7.5,\ 8])$. However, for some other 4-dimensional matrices, Algorithm 2.1(A) may not fall into some cycle. For example, if $A = diag(1, 5, 10, 15)$, we did not observe any cycle.

In the higher-dimension case, it is also possible that Algorithm 2.1(A) generates some kind of cycle though the possibility becomes smaller. Consider a 5-dimensional case with $A = diag([1,\ 3,\ 15,\ 28,\ 30])$. If the initial gradient is given by

$$g_1 = (6.805619838091368e\text{–}01,\ 2.052618801618283e\text{–}01,\ 8.364198798483330e\text{–}01,\\ 7.089206109330620e\text{–}01,\ 8.287079519725907e\text{–}01)^T,$$

which is obtained by the random number generator in MATLAB, then we observe that $g_k^{(3)}/\|g_k\|_2$ tends to zero and Algorithm 2.1(A) falls into a cycle similarly to the 4-dimension case. In most cases, we found that Algorithm 2.1(A) also produces some kind of cycle, but it is not stable. Figure 4.4 plots the trace of $\{\alpha^{-1}_{8i+1}\}$ of the algorithm with $g_1 = (0.1,\ 0.5,\ 0.3,\ 0.4,\ 1)^T$.

5. Some more efficient methods based on (2.10) or its variants

It is suggested in [11] to investigate other possibilities with the stepsize formula (2.10), such as using a stepsize (2.10) after every m exact line search iterations. In fact, both Algorithm 2.1(A) and Algorithm 2.1(B) are of such kind and takes a stepsize (2.10) after every 1 and 2 exact line search iterations, respectively. These methods have finite termination property for 2-dimensional convex quadratic due to

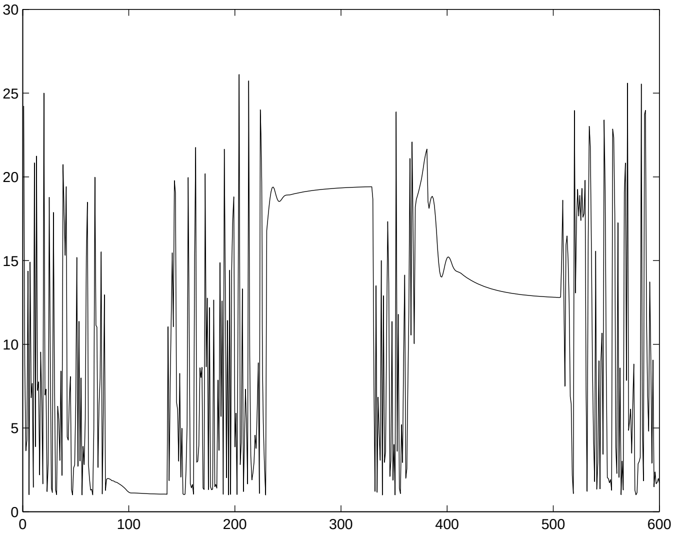

Figure 4.4: The subsequence $\{\alpha_{8i+1}^{-1}\}$ by Alg. 2.1(A) for a 5-d example

the proposition of (2.10) and the definition of these methods. However, we found that only the choice $m = 2$, which is corresponding to Algorithm 2.1(B), provides the property of "descreasing together". Meanwhile, the numerical results using the other choices are far more worse than those of Algorithm 2.1(B).

As will be shown, however, more efficient methods than Algorithm 2.1(B) can be easily obtained based on the formula (2.10) or its variants. One idea is to consider a larger class of gradient method that calculates the stepsize by (2.10) m_1 times after m_2 exact line search iterations. In this case, we give the following variant of (2.10):

$$\alpha_k^{YV} = \frac{2}{\sqrt{(1/\alpha_{k-1}^{SD} - 1/\alpha_k^{SD})^2 + 4\|g_k\|_2^2/(\alpha_{k-1}^{SD}\|g_{k-1}\|_2)^2} + 1/\alpha_{k-1}^{SD} + 1/\alpha_k^{SD}}. \quad (5.1)$$

The formula (5.1) is equivalent to (2.10) if $s_{k-1} = -\alpha_{k-1}^{SD} g_{k-1}$, namely, the previous iteration is obtained by an exact line search. Otherwise, if $\alpha_{k-1} \neq \alpha_{k-1}^{SD}$, they may be different. We consider the gradient methods with α_k given by

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,3) = 1, \\ \alpha_k^{YV}, & \text{otherwise} \end{cases} \quad (5.2)$$

and

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,4) = 1 \text{ or } 2, \\ \alpha_k^{YV}, & \text{otherwise,} \end{cases} \quad (5.3)$$

respectively. In other words, the above two methods calculate the stepsize by (5.1) twice after 1 and 2 exact line search iterations, respectively. It is easy to see that the two methods are still monotone. Meanwhile, with the same technique (3.6), we found that they also have the property of "descreasing together". In addition, the numerical results of the methods will become worse if α_k^{YV} is replaced by α_k^Y in the calculation of α_k.

Another idea of modifying 2.1(B) is to calculate a stepsize by (2.10) with k replaced by $k-1$ after every two SD iterations. Its basic motivation is that α_{k-2}^{SD} and α_{k-1}^{SD} are already computed and used in the previous two iterations. Mathematically, the stepsize has the form

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,3) \neq 0, \\ \alpha_{k-1}^{Y}. & \text{if } mod(k,3) = 0. \end{cases} \tag{5.4}$$

Since it is possible that $\alpha_{k-1}^Y > 2\alpha_k^{SD}$, the method (1.1)–(5.4) is not a monotone method any more. An analogue of the formula (5.4) is as follows:

$$\alpha_k = \begin{cases} \alpha_k^{SD2}, & \text{if } mod(k,3) \neq 0, \\ \alpha_{k-1}^{Y2}. & \text{if } mod(k,3) = 0, \end{cases} \tag{5.5}$$

where $\alpha_k^{SD2} = \frac{g_k^T A g_k}{g_k^T A^2 g_k}$ and

$$\alpha_k^{Y2} = \frac{2}{\sqrt{\left(\frac{1}{\alpha_{k-1}^{SD2}} - \frac{1}{\alpha_k^{SD2}}\right)^2 + \frac{4\, g_k^T A g_k}{(\alpha_{k-1}^{SD2})^2 g_{k-1}^T A g_{k-1}}} + \frac{1}{\alpha_{k-1}^{SD2}} + \frac{1}{\alpha_k^{SD2}}}. \tag{5.6}$$

Similarly to (2.10), the formula (5.6) has a property that, if $\alpha_1 = \alpha_1^{SD2}$, $\alpha_2 = \alpha_2^{Y2}$, and $\alpha_3 = \alpha_3^{SD2}$, then x_4 is the minimizer of any 2-dimensional convex quadratic function. Using the technique in (3.6), we found that the methods (5.4) and (5.5) also have the property of "descreasing together".

We tested the BB method, Algorithm 2.1(B), and the gradient methods (5.2), (5.3), (5.4) and (5.5) for a class of quadratic problems

$$f(x) = \frac{1}{2} x^T Diag(\lambda_1, \cdots, \lambda_n) x, \quad x \in \Re^n.$$

We consider three values 10^2, 10^3, 10^4 for both the dimension n and the condition number $Cond$. We let $\lambda_1 = Cond$, $\lambda_n = 1$ and $\lambda_i (i = 2, \cdots, n-1)$ be randomly generated in the interval $(1, \lambda_n)$. The starting point $x_1^{(i)} (i = 1, \cdots, n)$ are randomly generated in the interval $[-5, 5]$. The stopping condition is

$$\|g_k\|_2 \leq \varepsilon. \tag{5.7}$$

Four values 10^{-1}, 10^{-2}, 10^{-4} and 10^{-8} are used for ε so that a solution with different precision is obtained. For each case, 10 runs are made and the average numbers of iterations required by each algorithm are taken down in Table 4.5. The winner(s) for each problem is marked in bold.

n	$Cond$	ε	BB	2.1(B)	(5.2)	(5.3)	(5.4)	(5.5)
10^2	10^2	10^{-1}	42.7	**38.6**	**38.6**	42.1	40.2	39.4
		10^{-2}	59.8	54.9	**54.3**	55.0	56.4	55.2
		10^{-4}	88.9	80.8	83.7	**80.3**	85.1	83.7
		10^{-8}	147.8	139.3	140.9	134.1	136.8	**129.1**
10^2	10^3	10^{-1}	134.5	122.6	120.1	**105.8**	129.1	119.3
		10^{-2}	204.3	186.3	155.8	**143.9**	170.8	166.2
		10^{-4}	289.7	295.6	251.8	**221.4**	279.3	278.2
		10^{-8}	505.6	511.9	424.8	**376.9**	447.0	440.5
10^2	10^4	10^{-1}	422.8	470.3	361.8	**304.3**	402.2	413.5
		10^{-2}	628.9	670.6	538.3	**489.4**	573.4	590.9
		10^{-4}	964.8	1275.9	823.8	**727.3**	936.2	949.7
		10^{-8}	1509.9	2352.3	1475.8	**1243.1**	1614.7	1665.8
10^3	10^2	10^{-1}	51.2	**46.5**	49.2	49.8	48.3	47.3
		10^{-2}	66.2	**62.7**	62.8	64.9	**62.7**	63.9
		10^{-4}	94.4	93.0	95.6	**89.9**	92.4	92.2
		10^{-8}	147.6	150.8	148.1	**144.7**	147.9	147.2
10^3	10^3	10^{-1}	152.7	151.6	166.0	**132.1**	155.4	154.9
		10^{-2}	193.3	214.8	221.0	**180.7**	201.6	206.9
		10^{-4}	299.9	319.8	314.7	**280.0**	298.1	312.6
		10^{-8}	505.9	536.9	494.5	**437.3**	509.9	499.7
10^3	10^4	10^{-1}	515.7	557.1	540.6	**428.0**	464.4	497.7
		10^{-2}	673.0	865.0	719.7	**571.1**	651.5	676.9
		10^{-4}	1060.0	1423.4	1086.2	**800.4**	1007.7	1065.4
		10^{-8}	1609.7	2457.9	1750.7	**1412.4**	1819.5	1734.9
10^4	10^2	10^{-1}	59.3	55.8	**51.8**	55.8	55.1	54.5
		10^{-2}	72.2	68.7	68.2	71.0	67.6	**67.2**
		10^{-4}	100.3	97.1	**96.6**	100.7	97.9	98.1
		10^{-8}	152.6	152.8	**148.3**	150.8	151.6	149.9
10^4	10^3	10^{-1}	181.5	176.6	184.6	**157.7**	176.6	174.8
		10^{-2}	229.2	234.6	233.3	**205.0**	229.5	226.8
		10^{-4}	340.2	332.8	332.6	**293.5**	327.0	317.8
		10^{-8}	539.5	531.0	534.8	**471.4**	513.7	517.5
10^4	10^4	10^{-1}	570.4	614.5	567.8	**476.3**	520.8	580.7
		10^{-2}	797.9	881.3	781.1	**634.5**	767.8	818.0
		10^{-4}	1075.9	1408.0	1141.0	**928.9**	1207.6	1139.3
		10^{-8}	1635.1	2568.0	1667.4	**1546.4**	1995.7	1882.3

Table 4.5: Numerical comparisons of different gradient methods

From Table 4.5, we see that all the methods (5.2), (5.3), (5.4) and (5.5) perform better than Algorithm 2.1(B). Specifically, we can see that the method (5.3) is the winner for the most cases and uniformly better than the BB method. For the case that $\varepsilon = 10^{-8}$ and $Cond = 10^4$, the method (5.3) saves 11.04% of the iteration numbers in the average sense comparing with the BB method.

6. Conclusion and discussion

In this paper, we have introduced a new technique, (3.6), which enables a long-term observation for the gradient method. Our analyses show that one of the new algorithms, Algorithm 2.1(B), which is proposed by Yuan [11] recently, shares with the BB method the property that all the gradient components with respect to the eigenvectors of the function Hessian are decreasing together. Meanwhile, the other monotone gradient methods we tested do not possess this property. This might partly explain why Algorithm 2.1(B) is comparable to the BB method in numerical computations.

We have provided several numerical examples showing that the stepsizes generated by the AM method and the other algorithm by Yuan, Algorithm 2.1(A), may fall into cycles. However, we did not find any similar cycle for Algorithm 2.1(B) in numerical computations. Although all our analyses are numerical, some of the analyses, for example, the 4-dimensional example for Algorithm 2.1(A) could be strictly established in theory.

We have also found some more efficient gradient methods based on the formula (2.10) or its variants. Specifically, our numerical results show that the method (5.3), which is still monotone, is uniformly better than the BB method. Detailed/sound theoretical analysis on these alternatives or even other possibility need further studies. We feel that there is much room for obtaining some new formula even more efficient than (2.10). As the formula (2.10) is derived under the assumption that the step in the previous iteration is an exact line search step (SD step), we could do the same thing without this assumption. Namely, it would be possible to obtain a formula such as

$$\alpha_k^{New} = \Phi(\alpha_{k-1}, \alpha_k^{SD}, g_k, g_{k-1}).$$

Here α_{k-1} is the stepsize actually taken in the previous iteration and Φ is some function.

Additional results of the examples §4 are that, the AM method is only linearly convergent for 3-dimensional convex quadratics, and Algorithm 2.1(A) is only linearly convergent if $n = 4$. Noting that the sequence $\{\nu_k\}$ in (3.6) is the same as $\{\|g_k\|_2\}$, we may also use (3.6) to observe the convergence behavior of the gradient method. Assume that the function is provided by (1.3) and (3.1) with λ_i and g_1 given by (3.5). We observed that Algorithm 2.1(B), the method (5.2), and the method (5.3) are all superlinearly convergent in the case that $n = 3$. Further, it seems to us that the convergence of the method (5.2) is still superlinear even if $n = 4$. We do not know yet whether these results can be established or not in theory.

Acknowledgements. The first author of this paper is supported by the Alexander von Humboldt Foundation and the Chinese NSF grants 10171104 and 40233029. The second author is partly supported by the Chinese NSF grants 10231060.

References

[1] H. Akaike, *On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method*, Ann. Inst. Statist. Math. Tokyo, 11 (1959), pp. 1-17.

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

[3] A. Cauchy, *Méthode générale pour la résolution des systèms d'equations simultanées*, Comp. Rend. Sci. Paris, 25 (1847), pp. 46-89.

[4] Y. H. Dai and R. Fletcher, *On the Asymptotic Behaviour of some New Gradient Methods*, Numerical Analysis Report NA/212, Department of Mathematics, University of Dundee, 2003.

[5] Y. H. Dai and X. Q. Yang, *A New Gradient Method with an Optimal Stepsize Property*, Research report, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, 2001 .

[6] Y. H. Dai and Y. X. Yuan, *Alternate Minimization Gradient Method*, IMA Journal of Numerical Analysis, 23 (2003), pp. 377-393.

[7] H. C. Elman and G. H. Golub, *Inexact and preconditioned Uzawa algorithms for saddle point problems*, SIAM J. Numer. Anal., 31 (1994), pp. 1645-1661.

[8] R. Fletcher, *On the Barzilai-Borwein Method,* Research report, Department of Mathematics, University of Dundee, 2001.

[9] G. E. Forsythe, *On the asymptotic directions of the s-dimensional optimum gradient method*, Numerische Mathematik, 11 (1968), pp. 57-76.

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

[11] Y. Yuan, *A new stepsize for the steepest descent method*, Research report, Institute of Computational Mathematics and Scientific/Engineering Computing, Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences, 2004.