World Academy of Science, Engineering and Technology
International Journal of Mathematical and Computational Sciences
Vol:9, No:7, 2015

Steepest Descent Method with New Step Sizes

Bib Paruhum Silalahi, Djihad Wungguli, Sugi Guritman

***Abstract*—**Steepest descent method is a simple gradient method for optimization. This method has a slow convergence in heading to the optimal solution, which occurs because of the zigzag form of the steps. Barzilai and Borwein modified this algorithm so that it performs well for problems with large dimensions. Barzilai and Borwein method results have sparked a lot of research on the method of steepest descent, including alternate minimization gradient method and Yuan method. Inspired by previous works, we modified the step size of the steepest descent method. We then compare the modification results against the Barzilai and Borwein method, alternate minimization gradient method and Yuan method for quadratic function cases in terms of the iterations number and the running time. The average results indicate that the steepest descent method with the new step sizes provide good results for small dimensions and able to compete with the results of Barzilai and Borwein method and the alternate minimization gradient method for large dimensions. The new step sizes have faster convergence compared to the other methods, especially for cases with large dimensions.

***Keywords*—**Convergence, iteration, line search, running time, steepest descent, unconstrained optimization.

I. Introduction

OPTIMIZATION is the branch of applied mathematics which studies processes for obtaining the best decision which gives the maximum or minimum value of a function. Optimization problems can be categorized into the constraint optimization and the unconstraint optimization [5]. The optimization problem can be solved analytically or numerically. For the nonlinear unconstraint optimization problems with many variables, the issue is the problems are not able to be solved by analytical methods. We need a numerical method to solve these problems. In general, the numerical methods are iterative and one of them is the steepest descent method.

The steepest descent method was first introduced by Cauchy in 1847 [3], which is one of the most basic procedures to minimize differentiable function of several variables. In some cases, this method has slow convergence in leading to an optimal solution, this occurs because of the zigzag form of the steps. In recent years, it has been more apparent that an important issue of the steepest descent method is the selection of the step size. This selection may affect fast or slow of the convergence of a function to an optimal solution. Barzilai and Borwein [1] refined this method by modifying the step size and the performance of the results are pretty well for large dimension problems.

Bib Paruhum Silalahi and Sugi Guritman are lecturers at the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Bogor Agricultural University, Jalan Meranti, Kampus IPB Bogor 16680, Indonesia (e-mail: bibparuhum@gmail.com, guritman@yahoo.co.id).

Djihad Wungguli is with the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Bogor Agricultural University, Jalan Meranti, Kampus IPB Bogor 16680, Indonesia (e-mail: djihadwungguli@gmail.com).

Barzilai and Borwein results have sparked a lot of research on the method of steepest descent. Studies were conducted to obtain a step size that enables rapid convergence and monotonous. A related research is carried out by [4], called alternate minimization gradient method with the idea of combining the step size alternates between minimizing the value of the function and the norm along the line of steepest descent gradient. Another research was conducted by [7], with a new step size at even iterations and exact line search at odd iterations. Based on the studies that have been conducted, we modify the step size of the steepest descent method. Then we compare the results of our new step sizes algorithm against steepest descent method, Barzilai and Borwein method, the alternate minimization gradient method and Yuan method in terms of the iterations number and the running time.

II. Steepest Descent Method and Its Variants

Steepest descent (SD) method is a simple gradient method for the unconstrained optimization [2]:

$$\min_{\mathbf{x}\in R^n} f(\mathbf{x}), \tag{1}$$

where $f(\mathbf{x})$ is a continuous differential function in R^n. This iterative method has the following form:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k(-\mathbf{g}_k), \tag{2}$$

where $\mathbf{g}_k = \mathbf{g}(\mathbf{x}_k) = \nabla f(\mathbf{x}_k)$ is the gradient vector of $f(\mathbf{x})$ at the current iteration at point $\mathbf{x}_k$ and $\alpha_k > 0$ is the step size [6]. The step size α_k can be obtained with exact line search:

$$\alpha_k = argmin\{f(\mathbf{x}_k + a(-\mathbf{g}_k))\} \tag{3}$$

Searching the step size with exact line search (3) causes the slow convergence toward an optimal solution, this happens because the zigzag form of the steps. For quadratic functions optimization case, α_k with exact line search can be simplified to:

$$\alpha_k^{SD} = \frac{\mathbf{g}_k^T\mathbf{g}_k}{\mathbf{g}_k^T A\mathbf{g}_k} \tag{4}$$

where A is a Hessian matrix.

Steepest descent algorithm is as:

Step 0 Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and a tolerance limit $0 < \varepsilon < 1$. Set $k = 0$.
Step 1 Determine $\mathbf{g_k}$. If $\|\mathbf{g_k}\| \le \varepsilon$, stop.
Step 2 Determine α_k which minimizes $f(\mathbf{x}_k + \alpha_k\mathbf{d_k})$.
Step 3 Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{g_k}$.

```
Step 4   $k = k + 1$, and go to Step 1.
```

Determining of the step size in the steepest descent method has become an issue of concern. This step size determination may affect fast or slow convergence to the optimal solution. Several step sizes that have been studied are described in the following.

A. Barzilai and Borwein Method

Barzilai and Borwein (BB) gradient method [1] is a modification of the steepest descent method by changing the step size α_k. The main idea of Barzilai and Borwein method is the usage of the information in the previous iteration to determine the step size in the next iteration. The step size of this method is derived from the two-point approach to the secant equation based on the quasi-Newton method, namely:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - B_k^{-1}\mathbf{g}_k \tag{5}$$

where $B_k^{-1} = \alpha_k I$ and I is the identity matrix. With the Taylor series expansion for the quadratic approach, B_k can be determined by:

$$B_k = \arg\min_{B\in\mathbb{R}} \|B\boldsymbol{s}_{k-1} - \boldsymbol{y}_{k-1}\|^2 \tag{6}$$

or

$$B_k = \arg\min_{B^{-1}\in\mathbb{R}} \|\boldsymbol{s}_{k-1} - B^{-1}\boldsymbol{y}_{k-1}\|^2, \tag{7}$$

where $\boldsymbol{s}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$, and $\boldsymbol{y}_{k-1} = \mathbf{g}_k - \mathbf{g}_{k-1}$. From $B_k^{-1} = \alpha_k I$, (6) and (7), two step sizes are obtained:

$$\alpha_k^{BB1} = \frac{\mathbf{s}_{k-1}^T\mathbf{s}_{k-1}}{\mathbf{s}_{k-1}^T\mathbf{y}_{k-1}} \tag{8}$$

and

$$\alpha_k^{BB2} = \frac{\mathbf{s}_{k-1}^T\mathbf{y}_{k-1}}{\mathbf{y}_{k-1}^T\mathbf{y}_{k-1}} \tag{9}$$

Barzilai and Borwein algorithm is presented as in the following.

```
Step 0   Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and a tolerance limit
         $0 < \varepsilon < 1$. Set $k = 0$.
Step 1   Determine $\mathbf{g_k}$. If $\|\mathbf{g_k}\| \le \varepsilon$, stop.
Step 2   If $k = 0$ then specify $\alpha_0$ with exact line search. If not
         determine $\alpha_k$ with
             $\alpha_k = \frac{\mathbf{s}_{k-1}^T\boldsymbol{s}_{k-1}}{\mathbf{s}_{k-1}^T\boldsymbol{y}_{k-1}}$,   or   $\alpha_k = \frac{\mathbf{s}_{k-1}^T\boldsymbol{y}_{k-1}}{\boldsymbol{y}_{k-1}^T\boldsymbol{y}_{k-1}}$,
         a. where $\boldsymbol{s}_{k-1} = \mathbf{x_k} - \mathbf{x}_{k-1}$, and $\boldsymbol{y}_{k-1} = \mathbf{g}_k - \mathbf{g}_{k-1}$.
Step 3   Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{g}_k$.
Step 4   $k = k + 1$, and go to Step 1.
```

B. Alternate Minimization Gradient Method

In some sense, the principle for minimizing a continuous and twice differentiable function $f(\mathbf{x})$ is equivalent with minimizing the gradient norm $\|\mathbf{g}(\mathbf{x})\|$. That is the basic idea of alternate minimization gradient (AM) method [4]. This method is a modification of the steepest descent method that alternates the step size between minimizing the norm function value and gradient along the line of steepest descent. More precisely $k \ge 1$, we choose the step size so that

$$\alpha_{2k-1} = \arg\min_{\alpha\in\mathbb{R}}\{\|\mathbf{g}(\mathbf{x}_{2k-1} - \alpha\mathbf{g}_{2k-1})\|\} \tag{10}$$

and

$$\alpha_{2k} = \arg\min_{\alpha\in\mathbb{R}}\{f(\mathbf{x}_{2k} - \alpha\mathbf{g}_{2k})\}. \tag{11}$$

From (10) and (11), it can be obtained:

$$\alpha_k^{AM} = \begin{cases} \frac{\mathbf{g}_k^T A\mathbf{g_k}}{\mathbf{g}_k^T A^2\mathbf{g_k}}, & \text{if } k \text{ odd,} \\ \frac{\mathbf{g}_k^T\mathbf{g_k}}{\mathbf{g}_k^T A\mathbf{g_k}}, & \text{if } k \text{ even.} \end{cases} \tag{12}$$

Alternate minimization gradient algorithm is presented as in the following.

```
Step 0 Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and a tolerance limit
       $0 < \varepsilon < 1$. Set $k = 0$.
Step 1 Determine $\mathbf{g_k}$. If $\|\mathbf{g_k}\| \le \varepsilon$, stop.
Step 2 If k odd then assign $\alpha_k = \frac{\mathbf{g}_k^T A\mathbf{g_k}}{\mathbf{g}_k^T A^2\mathbf{g_k}}$.
       If not assign $\alpha_k = \frac{\mathbf{g}_k^T\mathbf{g_k}}{\mathbf{g}_k^T A\mathbf{g_k}}$.
Step 3 Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k\mathbf{g_k}$.
Step 4 $k = k + 1$, and go to Step 1.
```

C. Yuan Method

Yuan gradient method [7] uses the step sizes alternately as in the AM method. However, Yuan method uses a new step size. Yuan method uses the exact line search (4) on odd iterations, and then uses the following step size on even iterations:

$$\alpha_{2k}^Y = \frac{2}{\sqrt{(1/\alpha_{2k-1} - 1/\alpha_{2k})^2 + 4\|\mathbf{g}_{2k}\|^2/\|\mathbf{s}_{2k-1}\|^2} + 1/\alpha_{2k-1} + 1/\alpha_{2k}} \tag{13}$$

where $\mathbf{s}_{2k-1} = \mathbf{x}_{2k} - \mathbf{x}_{2k-1} = -\alpha_{2k-1}\mathbf{g}_{2k-1}$. In general the step size α_k of Yuan method is written as:

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } k \text{ odd} \\ \alpha_k^Y, & \text{if } k \text{ even,} \end{cases} \tag{14}$$

and Yuan algorithm is presented as in the following.

```
Step 0 Given a starting point $\mathbf{x}_1 \in \mathbb{R}^n$ and a tolerance limit
       $0 < \varepsilon < 1$.
Step 1 Determine $\mathbf{g_1}$ and $A$. If $\|\mathbf{g_1}\| \le \varepsilon$, stop. Set $k = 1$.
Step 2 Determine $\alpha_{2k-1}$. Then calculate $\mathbf{x}_{2k} = \mathbf{x}_{2k-1} - \alpha_{2k-1}\mathbf{g}_{2k-1}$.
Step 3 If $\|\mathbf{g_{2k}}\| \le \varepsilon$, stop.
Step 4 Determine $\alpha_{2k}$ and $\mathbf{s}_{2k-1} = \mathbf{x}_{2k} - \mathbf{x}_{2k-1}$. Then
       determine
           $\alpha_{2k}^Y = \frac{2}{\sqrt{(1/\alpha_{2k-1} - 1/\alpha_{2k})^2 + 4\|\mathbf{g}_{2k}\|^2/\|\mathbf{s}_{2k-1}\|^2} + 1/\alpha_{2k-1} + 1/\alpha_{2k}}$
       Assign $\mathbf{x}_{2k+1} = \mathbf{x}_{2k} - \alpha_{2k}^Y\mathbf{g_{2k}}$.
Step 5 If $\|\mathbf{g_{2k+1}}\| \le \varepsilon$, stop.
Step 6 Assign $k = k + 1$, and go to Step 2.
```

III. New Step Sizes

It has been discussed in the previous section that Yuan uses the step size with exact line search on odd iterations and then uses a step size (13) on even iterations as written in (14). Based on Yuan method, we modified steepest descent method by using two new step sizes. First we form an algorithm with the following steps:

$$\begin{aligned} \mathbf{x}_2 &= \mathbf{x}_1 - \alpha_1 \mathbf{g}_1 \\ \mathbf{x}_3 &= \mathbf{x}_2 - \alpha_2^Y \mathbf{g}_2 \\ \mathbf{x}_4 &= \mathbf{x}_3 - \alpha_3^Y \mathbf{g}_3 \\ \mathbf{x}_5 &= \mathbf{x}_4 - \alpha_4 \mathbf{g}_4 \end{aligned} \tag{15}$$

where α_1 and α_4 are the step sizes of the search process using the exact line search (4), whereas α_2^Y and α_3^Y using Yuan step size (13). Iteration process (15) is an early form of the whole iteration process, so the iteration process (15) will continue until a solution is found. In general α_k of the form (15) can be written as:

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,4) = 0 \text{ or } 1 \\ \alpha_k^Y, & \text{if } mod(k,4) = 2 \text{ or } 3 \end{cases} \tag{16}$$

New Step Size Algorithm 1

```
Step 0  Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and a tolerance limit $0 < \varepsilon < 1$. Set $k = 1$.
Step 1  Determine $\mathbf{g_k}$ dan $A$. If $\|\mathbf{g_k}\| \le \varepsilon$, stop.
Step 2  If $mod(k,4) = 0$ or 1 then
        $\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g_k}}{\mathbf{g}_k^T A \mathbf{g_k}}$, else determine $\mathbf{s}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$ and $a_k^Y = \frac{2}{\sqrt{(1/\alpha_{k-1} - 1/\alpha_k)^2 + 4\|\mathbf{g}_k\|^2/\|\mathbf{s}_{k-1}\|^2} + 1/\alpha_{k-1} + 1/\alpha_k}$.
Step 3  Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g_k}$
Step 4  $k = k + 1$, and go to Step 1.
```

For the second new step, we form algorithm with the following steps:

$$\begin{aligned} \mathbf{x}_2 &= \mathbf{x}_1 - \alpha_1 \mathbf{g}_1 \\ \mathbf{x}_3 &= \mathbf{x}_2 - \alpha_2 \mathbf{g}_2 \\ \mathbf{x}_4 &= \mathbf{x}_3 - \alpha_3^Y \mathbf{g}_3 \\ \mathbf{x}_5 &= \mathbf{x}_4 - \alpha_4^Y \mathbf{g}_4 \end{aligned} \tag{17}$$

where α_1 and α_2 are the step sizes using the exact line search (4), whereas α_3^Y and α_4^Y using Yuan step size (13). Similarly to the iteration process (15), the iteration in (17) also will continue until a solution is found. In general α_k of the form (17) can be written as:

$$\alpha_k = \begin{cases} \alpha_k^{SD}, & \text{if } mod(k,4) = 1 \text{ or } 2 \\ a_k^Y, & \text{if } mod(k,4) = 0 \text{ or } 3 \end{cases} \tag{18}$$

New Step Size Algorithm 2

```
Step 0  Given a starting point $\mathbf{x}_0 \in \mathbb{R}^n$ and a tolerance limit $0 < \varepsilon < 1$. Set $k = 1$.
Step 1  Determine $\mathbf{g_k}$ dan $A$. If $\|\mathbf{g_k}\| \le \varepsilon$, stop.
Step 2  If $mod(k,4) = 1$ or 2 then
        $\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g_k}}{\mathbf{g}_k^T A \mathbf{g_k}}$, else determine $\mathbf{s}_{k-1} = \mathbf{x}_k - \mathbf{x}_{k-1}$ and $a_k^Y = \frac{2}{\sqrt{(1/\alpha_{k-1} - 1/\alpha_k)^2 + 4\|\mathbf{g}_k\|^2/\|\mathbf{s}_{k-1}\|^2} + 1/\alpha_{k-1} + 1/\alpha_k}$.
Step 3  Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g_k}$
Step 4  $k = k + 1$, and go to Step 1.
```

IV. Numerical Results

This section presents comparison of numerical results of each method described in the previous section, namely SD, BB, AM, Yuan, algorithm 1 and algorithms 2. The step size of SD method uses (4) which is a simplification of the exact line search. BB method is divided into two parts, namely BB1 and BB2. BB1 uses step size (8) and BB2 uses step size (9). We compare the number of iterations and the running time of each method for obtaining the minimum value. We use nonlinear function with quadratic forms. Quadratic functions used in this study were generated randomly in the form of:

$$f(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T Diag(\lambda_1, \ldots, \lambda_n)(\mathbf{x} - \mathbf{x}^*), \quad \mathbf{x} \in \mathbb{R}^n,$$

with n = 2, 3, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100. Vector $\mathbf{x}_i^*$ $(i = 1, \cdots, n)$ is an integer random number in the interval [-5.5]. Furthermore $\lambda_1 = 1$ and $\lambda_n = 10{,}100{,}1000$, are the condition of the Hessian matrix of the function. Next $\lambda_i (i = 2, \cdots, n-1)$ is an integer random number in the interval $[1, \lambda_n]$. For all dimensions and the λ_n, the starting point is zero vector $(0, \cdots 0)^T$ and the stopping criteria is $\|\mathbf{g}_k\| \le 10^{-8}$. Experiments were performed 5 times for each dimension and each λ_n of each method, so that the experiments were carried out 105 times for each dimension. The total experiments for all dimensions were 1260 times. The average number of iterations and the running time are presented in Tables I and II.

Based on the results, in general it can be seen that the relationship between the dimensions, λ_n with the number of iterations and the running time. For the small dimension cases (n = 2 & 3), the value of λ_n does not give significant effect to the number of iterations and the running time. This means that the larger λ_n does not guarantee that the number of iterations and the running time will be larger (Tables I and II). For the larger dimensions (n = 10, 20, ⋯,100), it can be seen that the larger λ_n, the number of iterations and the running time will be larger (Figs. 1-3). Furthermore, we could not see that the larger dimension cause the larger iteration (Figs. 1 (a), 3 (a)). By contrast for the running time, it can be seen that the larger the dimension, the larger the running time become (Figs. 1 (b), 3(b)).

It can be seen that the Yuan method has found a solution of small dimension quadratic function problem with the smallest minimum number of iterations and the running time. Nevertheless the algorithm 1 and 2 are able to compete with Yuan method. For larger dimensions, Yuan method gives poor results, especially at $\lambda_n = 100$ and $\lambda_n = 1000$. By contrast to the algorithm 1 and 2, for large dimension cases with all sizes of λ_n, algorithm 1 and 2 is better in finding solutions in term of the number of iterations and the running time compared to BB methods, AM method and Yuan method (Figs. 1-3). We present the rate of convergence of each method in Fig. 4 for n = 100 dan λ_n = 100. We can see that the algorithm 1 and 2 have faster convergence rate compared to the others.

TABLE I
THE AVERAGE NUMBER OF ITERATIONS

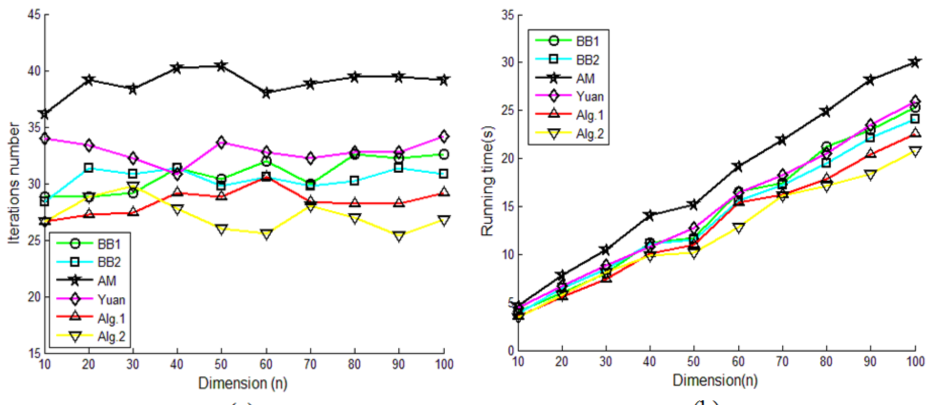
n	λ_n	The average number of iterations						
		SD	BB1	BB2	AM	Yuan	Alg.1	Alg.2
2	10	15.8	9.4	8.8	7.8	3.0	4.0	5.0
	100	17.6	10.2	9.4	8.0	3.0	4.0	5.0
	1000	18.0	9.6	8.2	6.6	3.0	4.0	5.0
3	10	58.0	16.0	15.4	21.6	15.0	12.4	14.8
	100	446.0	19.4	15.0	41.2	9.4	9.2	11.4
	1000	**	25.2	14.8	63.2	7.0	8.0	10.6
10	10	71.2	28.8	28.4	36.2	34.0	26.6	26.6
	100	572.2	73.4	59.0	82.8	77.2	55.4	50.8
	1000	**	84.4	62.4	177.6	215.8	66.4	49.8
20	10	69.8	28.8	31.4	39.2	33.4	27.2	28.8
	100	611.8	66.2	63.6	87.6	92.6	53.2	63.0
	1000	**	129.0	102.0	250.4	256.6	90.0	101.2
30	10	74.4	29.2	30.8	38.4	32.2	27.4	29.8
	100	618.0	81.8	84.0	94.8	119.6	67.2	78.4
	1000	**	161.2	90.4	214.0	372.2	117.6	96.2
40	10	71.0	31.4	31.4	40.2	30.8	29.2	27.8
	100	619.8	85.8	81.0	95.6	124.6	72.8	72.4
	1000	**	158.8	121.4	234.0	559.0	138.4	116.0
50	10	70.4	30.4	29.8	40.4	33.6	28.8	26.0
	100	568.4	83.0	81.4	92.6	105.8	69.6	73.0
	1000	**	169.6	127.2	256.0	741.4	115.2	123.6
60	10	73.0	32.0	30.6	38.0	32.8	30.6	25.6
	100	541.8	83.6	79.6	91.6	107.8	69.8	69.6
	1000	**	153.0	125.0	209.2	431.4	126.8	120.2
70	10	72.0	30.0	29.8	38.8	32.2	28.4	28.0
	100	588.3	98.2	84.2	103.2	114.6	77.4	81.8
	1000	**	174.8	122.0	321.0	514.2	127.8	129.2
80	10	73.4	32.6	30.2	39.4	32.8	28.2	27.0
	100	597.8	86.0	90.2	110.0	114.2	77.6	78.2
	1000	**	151.6	133.4	236.6	631.0	132.2	126.6
90	10	73.6	32.2	31.4	39.4	32.8	28.2	25.4
	100	671.2	94.4	87.8	105.0	127.8	79.8	80.6
	1000	**	160.0	147.0	249.6	676.8	128.0	133.0
100	10	73.2	32.6	30.8	39.2	34.2	29.2	26.8
	100	566.6	85.2	80.2	93.6	113.6	77.0	74.8
	1000	**	170.6	146.8	261.8	540.2	139.4	137.8

** More than 2000 iterations

TABLE II
THE AVERAGE RUNNING TIME

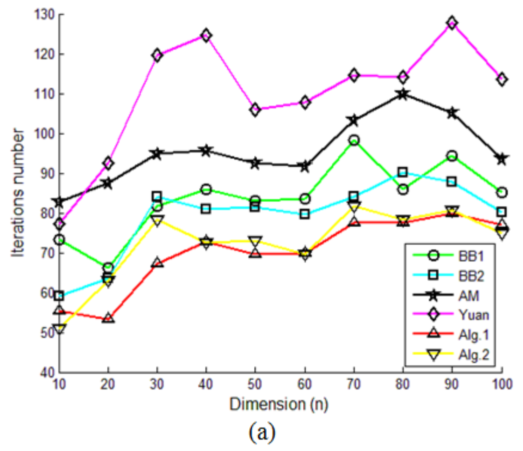
n	λ_n	The average running time						
		SD	BB1	BB2	AM	Yuan	1	2
2	10	1.2158	0.7921	0.7430	0.6681	0.3445	0.4234	0.4813
	100	1.3592	0.8915	0.7654	0.7447	0.3729	0.4249	0.5298
	1000	1.2957	0.7972	0.6808	0.5900	0.3510	0.4351	0.4789
3	10	4.1901	1.5578	1.2570	1.7168	1.2100	1.0527	1.1979
	100	38.2374	1.6251	1.1804	3.0742	0.8563	0.8072	0.9336
	1000	**	2.3408	1.2184	4.7670	0.6885	0.8337	0.9288
10	10	9.1579	4.0018	3.8183	4.6807	4.4170	3.5215	3.4583
	100	178.4230	9.9522	8.0304	10.7603	9.9536	7.0518	6.6856
	1000	**	10.0730	7.4047	20.6736	29.8327	7.4251	5.5743
20	10	13.9018	5.9759	6.5152	7.8569	6.7487	5.5918	5.7751
	100	161.2717	13.5476	13.0366	17.3494	18.5229	10.4792	12.3972
	1000	**	27.3802	20.9280	53.2933	61.3703	17.6327	20.4013
30	10	19.8876	8.0823	8.4538	10.5365	8.8325	7.4604	8.1181
	100	257.1545	23.5515	24.0258	27.0576	34.6199	18.6904	21.7436
	1000	**	50.7939	26.3331	64.7710	133.1709	33.0935	26.9119
40	10	24.1604	11.1630	11.0736	14.0317	10.7476	10.1022	9.8407
	100	298.8377	32.5804	30.0823	32.6756	42.8027	30.0220	25.4911
	1000	**	63.8277	46.8107	89.4547	282.7427	51.3329	41.6764
50	10	25.9427	11.7516	11.5382	15.2466	12.7024	11.0449	10.1426
	100	306.8892	34.1412	33.8299	36.8299	42.0616	27.4626	28.6959
	1000	**	83.3517	60.3106	121.2303	472.8374	49.5199	53.4718
60	10	36.3882	16.5356	15.6462	19.2288	16.4662	15.3770	12.7958
	100	363.0106	43.4831	41.6120	46.6732	54.9497	34.7719	34.9854
	1000	**	85.6454	67.6937	112.0454	271.2039	66.8134	60.9172
70	10	40.2148	17.4249	17.2778	21.9776	18.3037	16.1753	16.0799
	100	499.9960	58.9287	50.5146	58.8435	65.3898	43.6303	46.0443
	1000	**	118.4859	78.4548	140.2313	410.2627	75.4451	75.3167
80	10	45.6855	21.2567	19.5432	24.9432	20.5602	17.8875	17.1437
	100	560.9794	59.3693	62.4149	72.1113	74.9126	49.6631	50.0332
	1000	**	113.1994	96.2621	229.8008	517.2911	88.1097	84.3816
90	10	51.3049	22.9283	22.1720	28.2006	23.4393	20.4438	18.3391
	100	686.0457	73.2183	67.1283	77.4596	95.3471	57.6106	58.3811
	1000	**	138.6705	124.9331	206.0923	699.7920	98.0156	100.0168
100	10	55.3843	25.3197	24.0521	30.0698	25.9653	22.5381	20.8658
	100	603.5062	70.9071	66.3028	74.8516	91.7633	61.3530	59.6863
	1000	**	160.4586	133.0056	236.0443	603.7740	114.6884	116.0814

** More than 1800 second



(a) (b)

Fig. 1 Comparison of the average number of iterations (a) and the running time (b) of the steepest descent method variants for $\lambda_n = 10$



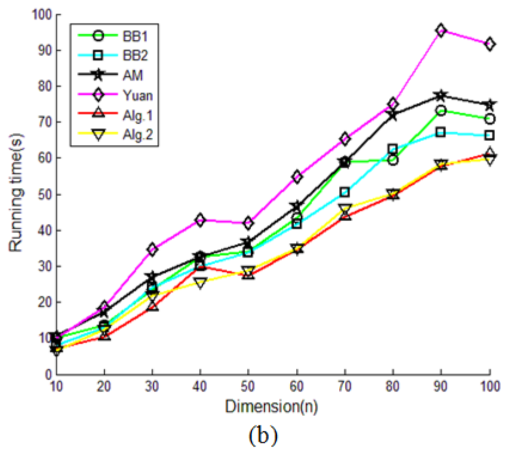
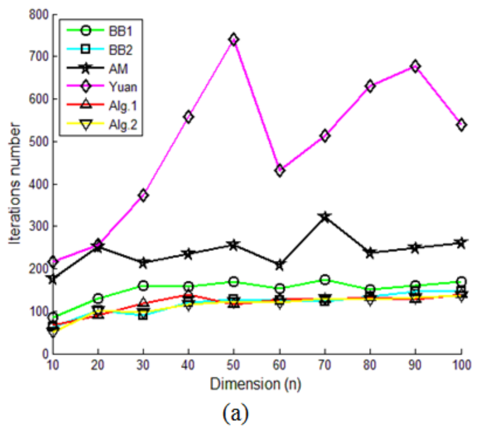


Fig. 2 Comparison of the average number of iterations (a) and the running time (b) of the steepest descent method variants for $\lambda_n = 100$



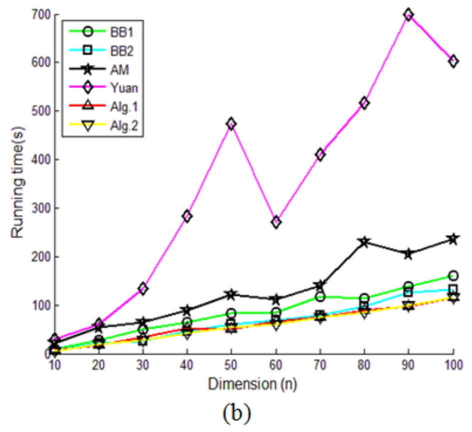


Fig. 3 Comparison of the average number of iterations (a) and the running time (b) of the steepest descent method variants for $\lambda_n = 1000$

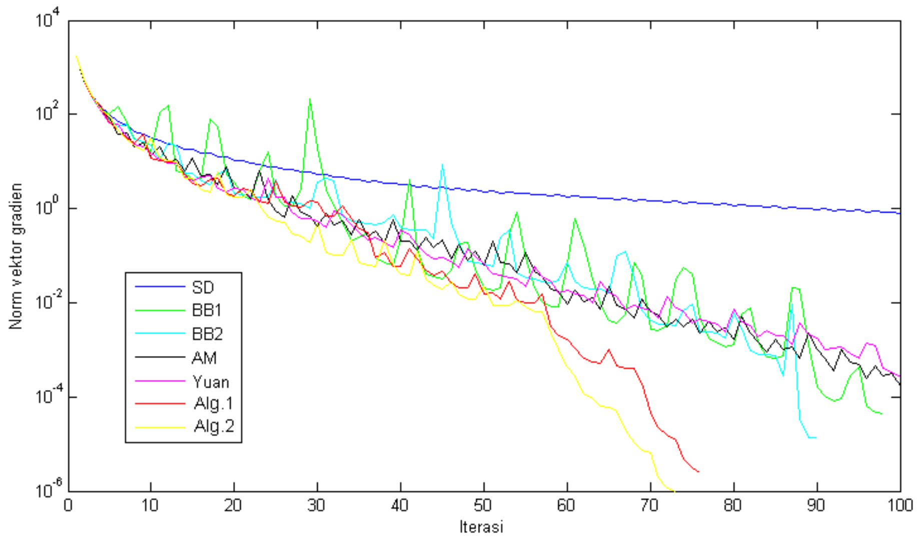


Fig. 4 The convergence rate of steepest descent method variants, for $n = 100$ dan $\lambda_n = 100$

V. Conclusion

Yuan method provides smaller number of iterations and the running time for small dimensions of quadratic functions case. However, Yuan method gives poor performance for the large dimensions and the large λ_n cases. We modified the step size of Yuan method. The new step size modification algorithms provide better performance for quadratic functions case with large dimensions or large λ_n compared to Yuan method. Even the new step size modification algorithms were able to surpass the performance of the Barzilai and Borwein method and the alternate minimization gradient method.

Acknowledgment

This work was supported by the Directorate General of Higher Education Ministry of National Education Indonesia (082/SP2H/PL/DIT.LITABMAS/ II/2015).

References

[1] Barzilai J, Borwein JM. 1988. Two point step size gradient methods. *IMA Journal of Numerical Analysis*, 8: 141-148.
[2] Bazara MS, Sherali HD, Shetty CM. 2006. *Nonlinear Programming: Theory and Algorithms*. USA: Wiley-Interescience.
[3] Cauchy A. 1847. General method for solving simultaneous equations Systems, *Comp. Rend. Sci. Paris,* 25: 46-89
[4] Dai YH, Yuan Y. 2003. Alternate minimization gradient method. *IMA Journal of Numerical Analysis*, 23: 377-393.
[5] Griva I, Nash SG, Sofer A. 2009. *Linear and Nonlinear Optimization*. USA: Society for Industrial and Applied Mathematics.
[6] Sun Wenyu, Yuan Y. 2006. *Optimization Theory and Methods: Nonlinear Programming*. New York: Spinger Science, Business Media.
[7] Yuan Y. 2006. A new stepsize for the steepest descent method. *Journal of Computational Mathematics*, 24:149-156.