Malaysian Journal of Mathematical Sciences 13(S) December: 65–75 (2019)
Special Issue: Conference on Mathematics, Informatics and Statistics (CMIS2018)

MALAYSIAN JOURNAL OF MATHEMATICAL SCIENCES

Journal homepage: http://einspem.upm.edu.my/journal

Comparison between BZAU, SRMI and MRM Conjugate Gradient Methods in Minimization Problems

Lai, L. Y.[1], Ibrahim, N. F. *[2], and Mohamed, N. A.[3]

[1]*School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Malaysia*
[2]*Marine Management Sciences Research Group, School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Malaysia*
[3]*Mathematics Department, Faculty of Science and Mathematics, Universiti Pendidikan Sultan Idris, Malaysia*

E-mail: fadhilah@umt.edu.my
* *Corresponding author*

ABSTRACT

The conjugate gradient method is one of the best methods that can be used to solve nonlinear unconstrained optimization problems. This method has gained the interest of researchers and has expanded rapidly. There are many versions of the conjugate gradient method. Each version claims to be efficient. In this paper, we make the comparison among three versions of the conjugate gradient method (MRM, SRMI and BZAU) by using exact line search. The methods were tested in terms of number of iteration and CPU time using 20 standard test functions. The result showed that MRM is the most efficient followed by BZAU and then SRMI. However, BZAU successfully found all the minimizers of the test functions whereas both SRMI and MRM failed at least once. In order to test the robustness of the methods, extensive tests are required.

Keywords: Conjugate gradient method; unconstrained; optimization; line search

1. Introduction

Consider the following unconstrained optimization problem:

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

where $f : R^n \to R$ is a continuously differentiable function. One of the most popular methods to solve this kind of problem is the conjugate gradient method. This is due to low memory requirement and global convergence properties.

Generally the method generated the sequence

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0, 1, 2, ..., \tag{1}$$

where the step length $\alpha_k > 0$ is obtained by some line search method. Both exact and inexact line searches can be used. For exact line search, the step length α_k is computed such that the objective function with d_k direction is exactly minimized. The formula is

$$\alpha_k = \min_{\alpha \geq 0} f(x_k + \alpha d_k). \tag{2}$$

One popular inexact line search used to find the step length is the Wolfe line search. The step length α_k is computed such that

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k g_k^T d_k, \tag{3}$$

$$g(x_k + \alpha_k d_k)^T d_k \geq \sigma g_k^T d_k, \tag{4}$$

where $0 < \rho < \sigma < 1$. The search direction d_k is generated by

$$d_k = \begin{cases} -g_k & if\ k = 0 \\ -g_k + \beta_k d_{k-1} & if\ k \geq 1 \end{cases} \tag{5}$$

where $g_k = \nabla f(x_k)$ is the gradient and β_k is a scalar. Most modifications of the conjugate gradient are on the β_k. Some of the well-known classic β_k are as follows.

Hestenes-Stiefel Formula (Hestenes and Stiefel (1952)),

$$\beta_k^{HS} = \frac{g_k^T (g_k - g_{k-1})}{d_{k-1}^T (g_k - g_{k-1})}$$

Polak-Ribiere-Polyak Formula (Polak and Ribiere (1969), Polyak (1969)),

$$\beta_k^{PRP} = \frac{g_k^T(g_k - g_{k-1})}{g_{k-1}^T g_{k-1}}$$

Fletcher-Reeves Formula (Fletcher and Reeves (1964)),

$$\beta_k^{FR} = \frac{g_k^T g_k}{g_{k-1}^T g_{k-1}}$$

Many versions of the conjugate gradient method can be found in literature. Each version claims to be efficient. It is important to determine which version is the most efficient among the new variety of conjugate gradient methods. In this paper, we make the comparison among three versions of conjugate gradient method, which are MRM (Hamoda et al. (2015)), SRMI (Shoid et al. (2015)) and BZAU (Baluch et al. (2017)).

In section 2, we will discuss the modified conjugate gradient method β_k^{MRM}, β_k^{SRMI} and β_k^{BZAU}. In section 3, we present the numerical results and discussion. Finally we conclude the findings in section 4.

2. The Methods

The three versions of conjugate gradient method that we compared are MRM, SRMI and BZAU.

The differences among versions of conjugate gradient method are usually step length α_k and parameter β_k. The MRM method used the following parameter β_k^{MRM}

$$\beta_k^{MRM} = \frac{g_k^T(g_k - \frac{\|g_k\|}{\|g_{k-1}\|} g_{k-1})}{\|g_{k-1}\|^2 + |g_{k+1}^T d_{k-1}|}$$

It was claimed to be promising and efficient when compared to Fletcher-Reeves (FR) and Polak-Ribiere-Polyak (PRP). The MRM method is globally converged under exact line search. The next method, SRMI, was proposed by Shoid et al. (2015). The method was modified from PRP and Shapiee et al. (2014) by taking the average β_k of the two methods.

$$\beta_k^{NRMI} = \frac{g_k^T(g_k - g_{k-1})}{g_{k-1}^T(g_k - d_{k-1})}$$

$$\beta_k^{SRMI} = PRP + NRMI$$

$$\beta_k^{SRMI} = \frac{\frac{g_k^T(g_k - g_{k-1})}{g_{k-1}^T g_{k-1}} + \frac{g_k^T(g_k - g_{k-1})}{g_{k-1}^T(g_k - d_{k-1})}}{2}$$

The convergence analysis showed that this method was globally converged under exact line search (Shoid et al. (2015)). When compared to FR, HS and RMIL methods, the SRMI method was more efficient.

The third method was BZAU which was introduced by Baluch et al. (2017). The method was modified from the PRP method and the method of Wei et al. (2006) denominator to produce the new parameter

$$\beta_k^{BZAU} = \frac{g_k^T(g_k - g_{k-1})}{-\eta g_{k-1}^T d_{k-1} + \mu \left|g_k^T d_{k-1}\right|},$$

for $\eta \in [1, +\infty)$, $\mu \in (\eta, +\infty)$. The best value for the parameter was $(\eta, \mu) = (1, 2)$. The algorithm proposed by Baluch et al. (2017) proved to be globally converged under Wolfe line search. The method also had sufficient descent property independent of any line search. Numerical result showed that the method outperformed TMPRP1 (Sun and Liu (2015))] which was more efficient than the CG_Descent method (Hager and Zhang (2006)) and DTPRP method (Dai and Wen (2012)).

The following is the general algorithm for the conjugate gradient method.

Algorithm 1:

Step 1: Given an initial point $x_0 \in R^n, \epsilon \geq 0$ and set $k = 0$, $d_0 = -g_0$if $\|g_0\| < \epsilon$, then stop.
Step 2: Compute α_k using line search.
Step 3: Let $x_{k+1} = x_k + \alpha_k d_k$, if $||g_{k+1}|| < \epsilon$, then stop.
Step 4: Compute β_k and d_{k+1}.
Step 5: Set $k = k + 1$ and go to Step 2.

2.1 Numerical Results and Discussion

Each of the three methods (MRM, SRMI and BZAU) claims to be more efficient. Each method was compared with different methods. Hence, we wanted to determine which method between MRM, SRMI and BZAU was the most efficient based on number of iterations and CPU time. The BZAU held sufficient descent property independent of any line search. Hence, in our experiment, we used exact line search and search direction (5). There were 20 test functions used. All the comparisons were done with three different initial points. All the test functions were solved by MATLAB software. In the experiment, we took $\epsilon = 10^{-6}$ and iteration was terminated when the stopping criteria $||g_k|| < 10^{-6}$ was fulfilled.

In the Tables 1 to 4, the symbol "FAIL" was represented when the routines of code stopped, since it failed to find the positive value of step size or when the number of iterations exceeded 1,000. Tables 1 and 2 show the performance comparison of MRM, SRMI and BZAU methods based on number of iterations. Tables 3 and 4 show the performance comparison of MRM, SRMI and BZAU methods based on CPU time.

From Tables 1 to 4, the BZAU method successfully reached the minimizer without fail. Each SRMI and MRM method failed to reach the minimizer at least once. From Table 5 and Table 6, the BZAU method outperformed SRMI, while MRM outperformed both BZAU and SRMI based on number of iteration. Table 5 shows that the BZAU method had 56.35% less number of iterations, 18.25% equal number of iterations and 25.40% greater number of iterations compared to SRMI. When compared to MRM, 34.13% of the BZAU had less number of iterations, 26.98% was equal and 38.89% of the BZAU had greater number of iterations. This also means that 38.89% of the MRM method had less number of iterations compared to BZAU. Table 6 also shows that 52.38% of MRM method had less number of iterations compared to SRMI.

From Table 7 and 8, BZAU outperformed both SRMI and MRM method, while MRM outperformed SRMI in terms of CPU time. From Table 7, 67.46% of the BZAU method had less CPU time compared to SRMI method and 53.17% of the BZAU method had less CPU time compared to the MRM method. From Table 8, 66.67% of the MRM method had less CPU time compared to the SRMI method.

Therefore, we can conclude that MRM outperformed BZAU and SRMI.

Table 1: Performance comparison between MRM, SRMI and BZAU based on number of iterations.

No	Function	Initial Point	BZAU	SRMI	MRM
1.	Rosenbrock (n=3)	(5,5,5) (-10,-10,-10) (11.5,11.5,11.5)	107 379 155	72 96 81	39 126 380
2.	Rosenbrock (n=5)	(5,...,5) (-10,...,-10) (11.5,...,11.5)	121 241 306	298 135 234	208 FAIL 403
3.	Rosenbrock(n=8)	(5,...,5) (-10,...,-10) (11.5,...,11.5)	263 184 259	315 385 427	101 FAIL 418
4.	Dixon-Price (n=3)	(2,2,2) (3,3,3) (10,10,10)	25 17 46	24 24 45	20 25 54
5.	Dixon-Price (n=5)	(2,...,2) (3,...,3) (10,...,10)	34 50 47	39 39 105	39 40 37
6.	Dixon-Price (n=8)	(2,...,2) (3,...,3) (10,...,10)	63 46 58	62 59 FAIL	59 53 FAIL
7.	Schwefel (n=3)	(330,330,330) (440,440,440) (550,550,550)	3 3 3	2 2 2	2 2 2
8.	Schwefel (n=5)	(330,...,330) (440,...,440) (550,...,550)	3 3 3	2 2 2	2 2 2
9.	Schwefel (n=8)	(330,...,330) (440,...,440) (550,...,550)	3 3 3	2 2 2	2 2 2
10.	Levy(n=3)	(2,2,2) (-1,-1,-1) (-2,-2,-2)	20 11 20	17 16 20	12 16 31
11.	Levy(n=5)	(2,...,2) (-1,...,-1) (-2,...,-2)	25 22 20	15 16 21	22 11 23
12.	Levy(n=8)	(2,...,2) (-1,...,-1) (-2,...,-2)	27 16 21	16 13 21	23 11 24
13.	Zakharov (n=2)	(3,3) (5,5) (8,8)	4 4 5	4 4 4	5 5 4
14.	Zakharov (n=3)	(3,3,3) (5,5,5) (8,8,8)	4 5 5	5 7 11	5 10 8
15.	Zakharov (n=4)	(3,3,3,3) (5,5,5,5) (8,8,8,8)	5 10 6	14 13 11	8 6 9
16.	Booth(n=2)	(3,3) (5,5) (8,8)	4 5 4	8 5 5	5 5 5
17.	Trid (n=3)	(5,5,5) (10,10,10) (15,15,15)	4 3 4	9 11 12	4 4 4
18.	Trid (n=5)	(5,...,5) (10,...,10) (15,...,15)	8 7 8	19 17 20	7 7 6
19.	Trid (n=8)	(5,...,5) (10,...,10) (15,...,15)	17 12 9	33 28 25	8 12 10
20.	Rotated hyper-ellipsoid (n=3)	(5,5,5) (10,10,10) (15,15,15)	6 6 6	11 12 12	6 6 6
21.	Rotated hyper-ellipsoid (n=5)	(5,...,5) (10,...,10) (15,...,15)	10 11 11	16 16 16	9 10 10
22.	Rotated hyper-ellipsoid (n=8)	(5,...,5) (10,...,10) (15,...,15)	15 16 16	22 23 23	14 14 14
23.	Sum of Different Power (n=3)	(-1,-1,-1) (2,2,2) (3,3,3)	16 36 17	149 98 76	20 23 26
24.	Sum of Different Power (n=5)	(-1,...,-1) (2,...,2) (3,...,3)	22 38 41	894 43 269	32 36 51
25.	Sum of Different Power (n=8)	(-1,...,-1) (2,...,2) (3,...,3)	273 79 33	227 465 543	41 35 41
26.	Beale (n=2)	(0,0) (2,2) (3,4)	9 9 9	17 20 19	9 14 17
27.	Colville (n=4)	(3,3,3,3) (5,5,5,5) (8,8,8,8)	318 194 293	402 215 271	170 234 245

Table 2: Performance comparison between MRM, SRMI and BZAU based on number of iterations.

28.	Styblinski-Tang (n=3)	(-1,-1,-1) (-3,-3,-3) (5,5,5)	2 4 2	5 2 4	2 2 6
29.	Styblinski-Tang (n=5)	(-1,...,-1) (-3,...,-3) (5,...,5)	4 4 5	2 2 3	2 3 3
30.	Styblinski-Tang (n=8)	(-1,...,-1) (-3,...,-3) (5,...,5)	2 4 4	3 3 3	2 3 4
31.	Sum squares (n=3)	(3,3,3) (5,5,5) (8,8,8)	6 6 6	10 11 11	6 6 6
32.	Sum square (n=5)	(3,...,3) (5,...,5) (8,...,8)	10 10 11	15 16 16	9 9 10
33.	Sum squares (n=8)	(3,...,3) (5,...,5) (8,...,8)	15 15 15	22 22 22	13 14 14
34.	Sphere (n=3)	(3,3,3) (5,5,5) (8,8,8)	2 2 2	2 2 2	2 2 2
35.	Sphere (n=5)	(3,...,3) (5,...,5) (8,...,8)	2 2 2	2 2 2	2 2 2
36.	Sphere (n=8)	(3,...,3) (5,...,5) (8,...,8)	2 2 2	2 2 2	2 2 2
37.	Modified sphere (n=6)	(3,...,3) (5,...,5) (8,...,8)	8 3 8	32 32 34	8 8 8
38.	Three-Hump Camel (n=2)	(1,1) (-3,-3) (5,5)	5 5 5	9 7 9	8 5 7
39.	Six-Hump Camel (n=2)	(1,1) (-1,-1) (3,3)	5 5 6	5 5 8	10 8 10
40.	Bohachevsky (n=2)	(1.5,1.5) (5,5) (9.5,9.5)	6 7 7	8 7 10	10 11 9
41.	Schaffer N2 (n=2)	(5,5) (10,10) (20,20)	1 1 1	1 1 1	1 1 1
42.	Matyas (n=2)	(2,2) (5,5) (8,8)	1 1 1	1 1 1	1 1 1

Table 3: Performance comparison between MRM, SRMI and BZAU based on CPU time.

No	Function	Initial Point	BZAU	SRMI	MRM
1.	Rosenbrock (n=3)	(5,5,5)	38.5000	23.9948	13.5469
		(-10,-10,-10)	134.6094	32.0208	41.8906
		(11.5,11.5,11.5)	51.6615	27.5104	137.1250
2.	Rosenbrock (n=5)	(5,...,5)	68.6458	165.9635	112.7135
		(-10,...,-10)	132.1458	72.4323	FAIL
		(11.5,...,11.5)	171.4479	130.2396	233.5104
3.	Rosenbrock(n=8)	(5,...,5)	263.1354	322.7396	95.9948
		(-10,...,-10)	179.4010	402.5521	FAIL
		(11.5,...,11.5)	258.2813	451.8281	439.1615
4.	Dixon-Price (n=3)	(2,2,2)	9.0729	8.78125	7.5729
		(3,3,3)	6.8646	8.9531	9.0156
		(10,10,10)	16.8385	15.4583	18.1927
5.	Dixon-Price (n=5)	(2,...,2)	18.6667	21.7240	21.3385
		(3,...,3)	28.0469	21.3281	21.8438
		(10,...,10)	25.2552	56.3490	20.6250
6.	Dixon-Price (n=8)	(2,...,2)	60.4010	59.5469	56.8490
		(3,...,3)	44.5625	56.8438	50.8385
		(10,...,10)	55.9375	FAIL	FAIL
7.	Schwefel (n=3)	(330,330,330)	2.0156	1.6406	1.5938
		(440,440,440)	2.3438	1.7031	1.5729
		(550,550,550)	1.9271	1.7083	1.5885
8.	Schwefel (n=5)	(330,...,330)	2.8385	2.2552	2.1042
		(440,...,440)	2.9271	2.3385	1.9635
		(550,...,550)	3.0104	2.5469	2.1771
9.	Schwefel (n=8)	(330,...,330)	4.1823	3.2344	3.4583
		(440,...,440)	4.2760	3.3359	3.3434
		(550,...,550)	4.2344	3.3073	3.5729
10.	Levy(n=3)	(2,2,2)	7.4531	6.5938	5.2031
		(-1,-1,-1)	4.9375	6.6094	7.6094
		(-2,-2,-2)	7.6094	7.5938	11.4063
11.	Levy(n=5)	(2,...,2)	14.2500	9.2344	12.7031
		(-1,...,-1)	13.7500	10.5000	7.2031
		(-2,...,-2)	12.0313	12.0313	13.1094
12.	Levy(n=8)	(2,...,2)	26.2344	16.2031	23.2969
		(-1,...,-1)	17.7188	14.9063	12.7813
		(-2,...,-2)	21.6563	21.6563	24.0781
13.	Zakharov (n=2)	(3,3)	1.5000	1.5000	1.7031
		(5,5)	2.9688	1.6875	1.6563
		(8,8)	1.9844	1.8906	1.3594
14.	Zakharov (n=3)	(3,3,3)	2.0469	2.0625	2.0569
		(5,5,5)	2.7188	2.8438	4.2188
		(8,8,8)	2.5625	4.7188	3.0781
15.	Zakharov (n=4)	(3,3,3,3)	2.6250	6.4844	4.0156
		(5,5,5,5)	5.0469	5.7813	3.1875
		(8,8,8,8)	3.1250	5.4219	4.2344
16.	Booth(n=2)	(3,3)	1.3438	2.4219	1.5625
		(5,5)	1.5469	1.5938	1.6250
		(8,8)	1.3438	1.7500	1.6875
17.	Trid (n=3)	(5,5,5)	2.2813	4.4844	2,2969
		(10,10,10)	1.8594	4.5781	2.2344
		(15,15,15)	2.1094	5.0625	2.4531
18.	Trid (n=5)	(5,...,5)	5.5000	12.1406	5.1875
		(10,...,10)	4.9375	10.5156	4.8281
		(15,...,15)	7.5469	11.9531	4.2031
19.	Trid (n=8)	(5,...,5)	18.1719	35.3438	10.2656
		(10,...,10)	13.0625	29.2969	14.1563
		(15,...,15)	10.2031	25.7500	11.6563
20.	Rotated hyper-ellipsoid (n=3)	(5,5,5)	4.2031	5.1250	4.8906
		(10,10,10)	4.0625	5.1563	3.6875
		(15,15,15)	4.2344	5.1563	3.0938
21.	Rotated hyper-ellipsoid (n=5)	(5,...,5)	7.0781	9.6875	6.3125
		(10,...,10)	7.3750	10.4219	7.1406
		(15,...,15)	7.3281	10.2500	6.6563
22.	Rotated hyper-ellipsoid (n=8)	(5,...,5)	18.6250	23.6719	15.3125
		(10,...,10)	17.1094	24.4531	26.5938
		(15,...,15)	17.0781	24.6250	24.6563
23.	Sum of Different Power (n=3)	(-1,-1,-1)	6.8594	51.5313	7.3906
		(2,2,2)	14.9844	32.4375	8.4375
		(3,3,3)	16.9063	25.1250	9.2500
24.	Sum of Different Power (n=5)	(-1,...,-1)	14.9844	629.0000	17.4531
		(2,...,2)	21.8438	24.5469	19.6875
		(3,...,3)	27.0313	152.0313	27.1406
25.	Sum of Different Power (n=8)	(-1,...,-1)	291.8594	222.7188	39.8594
		(2,...,2)	78.3438	491.9063	34.5625
		(3,...,3)	31.8438	586.6406	39.3125
26.	Beale (n=2)	(0,0)	3.4375	5.4219	3.4375
		(2,2)	3.8594	6.3434	4.7969
		(3,4)	3.8906	6.0625	5.3281
27.	Colville (n=4)	(3,3,3,3)	144.5313	189.3125	75.5000
		(5,5,5,5)	84.6406	99.8750	102.5313
		(8,8,8,8)	133.9688	121.3281	107.8594

Table 4: Performance comparison between MRM, SRMI and BZAU based on CPU time

28.	Styblinski-Tang (n=3)	(-1,-1,-1) (-3,-3,-3) (5,5,5)	1.6406 2.3125 1.5469	2.5156 1.8281 2.3750	1.5313 1.4531 3.1094
29.	Styblinski-Tang (n=5)	(-1,...,-1) (-3,...,-3) (5,...,5)	3.4844 3.6406 4.1875	2.1719 2.6719 2.7969	2.3750 2.7656 2.7813
30.	Styblinski-Tang (n=8)	(-1,...,-1) (-3,...,-3) (5,...,5)	3.3125 5.4219 5.3594	4.5156 4.4219 4.5156	3.9531 4.5156 5.5156
31.	Sum squares (n=3)	(3,3,3) (5,5,5) (8,8,8)	2.6563 2.8906 2.8906	4.1875 4.6406 4.6094	3.2500 3.2031 3.0313
32.	Sum square (n=5)	(3,...,3) (5,...,5) (8,...,8)	6.2188 6.5000 7.2656	9.1094 9.9844 9.6875	6.1250 6.2031 6.7031
33.	Sum squares (n=8)	(3,...,3) (5,...,5) (8,...,8)	16.4688 16.4063 16.0469	24.0938 23.4063 23.2813	15.4688 16.3594 15.5781
34.	Sphere (n=3)	(3,3,3) (5,5,5) (8,8,8)	1.3906 1.5156 1.4844	1.7813 1.7500 1.7344	1.5625 1.6719 1.6875
35.	Sphere (n=5)	(3,...,3) (5,...,5) (8,...,8)	2.0313 2.2813 2.2656	2.0625 2.0469 2.0156	2.2031 2.3125 2.0469
36.	Sphere (n=8)	(3,...,3) (5,...,5) (8,...,8)	3.5156 3.4219 3.3125	3.6094 3.3906 3.4219	3.3594 3.4063 3.5000
37.	Modified sphere (n=6)	(3,...,3) (5,...,5) (8,...,8)	7.3906 6.5469 6.5781	23.0000 24.8438 25.6094	6.5937 7.3750 6.7656
38.	Three-Hump Camel (n=2)	(1,1) (-3,-3) (5,5)	2.1563 2.0938 2.0313	3.0313 2.7188 3.1406	2.8125 1.9375 2.5156
39.	Six-Hump Camel (n=2)	(1,1) (-1,-1) (3,3)	2.1250 1.8750 2.2031	2.1875 1.9844 2.7031	3.3438 2.9063 3.5469
40.	Bohachevsky (n=2)	(1.5,1.5) (5,5) (9.5,9.5)	2.2188 2.5313 2.6094	2.6719 2.6719 3.5000	3.5469 3.5625 2.9531
41.	Schaffer N2 (n=2)	(5,5) (10,10) (20,20)	2.1563 2.0938 2.0313	3.0313 2.7188 3.1406	2.8125 1.9375 2.5156
42.	Matyas (n=2)	(2,2) (5,5) (8,8)	0.9688 1.0156 0.7969	0.9375 0.8750 0.8125	1.3125 1.1719 1.1563

Table 5: Percentage performance of BZAU method compared to SRMI and MRM based on number of iterations.

Method		Comparison	
		SRMI	MRM
BZAU	Less number of iterations	56.35%	34.13%
	Equal number of iterations	18.25%	26.98%
	Greater number of iterations	25.40%	38.89%

Table 6: Percentage performance of MRM method compared to BZAU and SRMI based on number of iterations.

Method		Comparison	
		BZAU	SRMI
MRM	Less number of iterations	38.89%	52.38%
	Equal number of iterations	26.98%	27.78%
	Greater number of iterations	34.13%	19.84%

Table 7: Percentage performance of BZAU method compared to SRMI and MRM based on CPU time.

Method		Comparison	
		SRMI	MRM
BZAU	Less CPU time	67.46%	53.17%
	Equal CPU time	2.38%	0.79%
	Greater CPU time	30.16%	46.03%

Table 8: Percentage performance of MRM method compared to BZAU and SRMI based on CPU time.

Method		Comparison	
		BZAU	SRMI
MRM	Less CPU time	46.03%	66.67%
	Equal CPU time	0.79%	0.79%
	Greater CPU time	53.17%	32.54%

3. Conclusions

In this paper, three different versions of conjugate gradient which were BZAU method, SRMI method and MRM method were compared using 20 standard test functions. The methods were compared in terms of number of iteration and CPU time in order to determine the efficiency of the method. The result showed that MRM was the most efficient followed by BZAU and then SRMI. However, BZAU successfully found all the minimizers of the test functions. Both SRMI and MRM failed at least once. In this test, however, only exact line search and search direction (5) was used. If different line search and different search directions were used, the result could be different. Furthermore, we only used 20 test functions. In order to test the robustness of the methods, extensive testing is required.

Acknowledgement

We acknowledge the support from Research Acculturation Grant Scheme (RAGS) research grant by the Ministry of Higher Education, Malaysia (MOHE).

References

Baluch, B., Salleh, Z., Alhawarat, A., and Roslan, U. A. M. (2017). A new modified three-term conjugate gradient method with sufficient descent property and its global convergence. *Journal of Mathematics*.

Dai, Z. and Wen, F. (2012). Another improved wei-yao-liu nonlinear conjugate gradient method with sufficient descent property. *Applied Mathematics and Computation*, 218(14):7421–7430.

Fletcher, R. and Reeves, C. M. (1964). Function minimization by conjugate gradients. *The computer journal*, 7:149–154.

Hager, W. W. and Zhang, H. (2006). A survey of nonlinear conjugate gradient methods. *Pacific Journal of Optimization. An International Journal*, 2(1):35–58.

Hamoda, H., Rivaie, M., Mamat, M., and Salleh, Z. (2015). A new nonlinear conjugate gradient coefficient for unconstrained optimization. *Applied Mathematical Sciences*, 9(37):1813 – 1822.

Hestenes, M. R. and Stiefel, E. (1952). Method of conjugate gradient for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436.

Polak, E. and Ribiere, G. (1969). Note sur la convergence de directions conjuge'es. *Rev. Francaise Informat Recherche Operationelle*, pages 35–43.

Polyak, B. T. (1969). The conjugate gradient method in extreme problems. *USSR Computational Mathematics and Mathematical Physics*, 9:94–112.

Shapiee, N., Mamat, R. M., and Mohd, I. (2014). A new modification of hestenesstiefel method with descent properties. *AIP Conference Proceedings*, pages 520–526.

Shoid, S., Rivaie, M., Mamat, M., and Salleh, Z. (2015). A new conjugate gradient method with exact line search. *Applied Mathematical Sciences*, 9(96):4799 – 4812.

Sun, M. and Liu, J. (2015). Three modified polak-ribiere-polyak conjugate gradient methods with sufficient descent property. *Journal of Inequalities and Applications*, (1).

Wei, Z., Li, G., and Qi, L. (2006). New nonlinear conjugate gradient formulas for large-scale unconstrained optimization problems. *Applied Mathematics and Computation*, 179(2):407–430.