Benchmarking of Optimization Algorithms with Wood's Function

Şeyma BAŞARAN
Dept. of Mathematical Engineering
Yıldız Technical University
İstanbul, Türkiye
seyma.basaran1@std.yildiz.edu.tr

Prof. Hale KÖÇKEN
Dept. of Mathematical Engineering
Yıldız Technical University
İstanbul, Türkiye
hgonce@yildiz.edu.tr

Lect. PhD Gökhan GÖKSU
Dept. of Mathematical Engineering
Yıldız Technical University
İstanbul, Türkiye
gokhan.goksu@yildiz.edu.tr

***Abstract*— In this study, five different algorithms are tested to optimize the Wood function. Each of the algorithms exhibited different approaches to iteratively optimize the Wood function. The behavior of the algorithms with respect to initial conditions is analyzed in detail. Furthermore, the performance of each algorithm is analyzed by comparing the number of iterations and execution times of the algorithms.**

***Index Terms*—optimization, algorithm, benchmarking, gradient, Wood's**

I. Introduction

Optimization problems are frequently encountered in engineering applications. Innumerous applications it is required to detect the globally optimal solutions. Economics, finance, networks and mechanical design, molecular biology, image processing and chemical engineering are just some of the scientific fields where such applications can bemet.[1] As the number of optimization methods, and implementations of those methods, has increased, researchers have pursued comparative studies to evaluate their performance. When done well, such studies can be of great value in helping end-users choose the most suitable optimization method for their problems. Such studies are generally referred to as optimization benchmarking.[2]

In the most general sense, benchmarking is the comparison of one or more products to an industrial standard product over a series of performance metrics. In the case of benchmarking optimization algorithms, the products are the specific implementations of given algorithms, and the performance metrics are generated by running the implementations on a series of test problems. This framework presents a certain clarity in benchmarking optimization algorithms, as there is at least some agreement on what constitutes "better".[2]

This paper proposes a method to calculate the minimum point of the Wood function in order to compare the performance of different optimization algorithms. The results obtained using Newton-Raphson, Hestenes-Stiefel, Polak-Ribière, Fletcher-Reeves and Dai-Yuan algorithms are analyzed in terms of the number of iterations, convergence speed and total running time. This analysis provides valuable information for the selection of algorithms for complex optimization problems.

II. The Problem

The problem consists of the Wood's function subject to $-10 \leq x_i \leq 10$ where $i \in \{1,2,3,4\}$.

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1) \quad (1)$$

The only minimum is located at $x^* = (1,1,1,1)$ with $f(x^*)=0$. The function has a saddle near (1,1,1,1). And also functions is continuous, differentiable.

III. Implementation

Algorithms and benchmarking code are implemented in MATLAB. Initial points are selected from uniform distribution $x_{1,2,3,4} \sim U_4(-10,10)$ from the problem domain interval, using the rand() MATLAB function. Error bound is taken as $\varepsilon = 10^{-4}$ and the stopping criterions are

$$\|\nabla f(x_k)\| \leqslant \varepsilon \text{ and } |f(x_{k+1}) - f(x_k)| \leqslant \varepsilon.$$

IV. Results

Execution times and number of iterations play an important role in evaluating the efficiency and effectiveness of optimization algorithms. In this project, we used five algorithms, Newton-Raphson, Hestenes-Stiefel, Polak-Ribiere, Fletcher-Reeves and Dai-Yuan, to solve optimization problems in MATLAB environment.

When solving the problem, the other algorithms we used, except Newton, had to use the alpha for the gradient calculation. However, in many trials, the minimum result of the alpha with the desired range and function coefficients in the original solution of the problem was always 0.

The complexity of this equation makes iterative methods impractical due to computational limitations. Therefore, for the purposes of this analysis, this report presents the original problem with the coefficients of the equation set to 100=>10, 90=>9, 10.1=>1 and 19.8=>1.98. This refined formulation provides a more systematic and feasible approach to solve the equation numerically.

A. First Initial Guess:

Algorithms were tested for the first initial guess as

$$x_0 = [0.5085 \quad 0.5108 \quad 0.8176 \quad 0.7948]$$

1) Also, the stopping criteria is taken as $\|\nabla f(x_k)\| \leq \varepsilon$:

I. Performance

Algorithm Name	Number of Iterations	Elapsed Time	Min. value	Error
Newton-Raphson	10	0.794321	0.000000	0.000002
Hestenes-Stiefel	50	7.312128	0.000000	0.000017
Polak-Ribiere	34	4.908347	0.000000	0.000093

Fletcher-Reeves	17	1.536889	0.003832	0.000000
Dai-Yuan	55	7.785370	0.000000	0.000080

As can be seen in Table I Newton-Raphson, Hestenes-Stiefel, Polak-Ribiere and Dai-Yuan algorithms are optimized algorithms since they give the results minimum. However, the Newton-Raphson algorithm has a shorter execution time, which makes it better than the others.

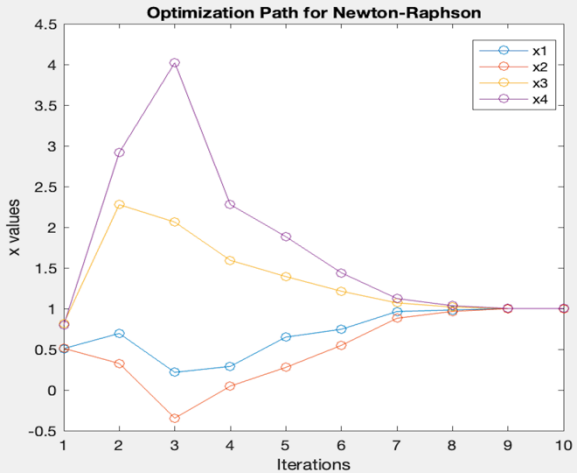

Fig. 1. Convergence path of each x values for Newton Raphson

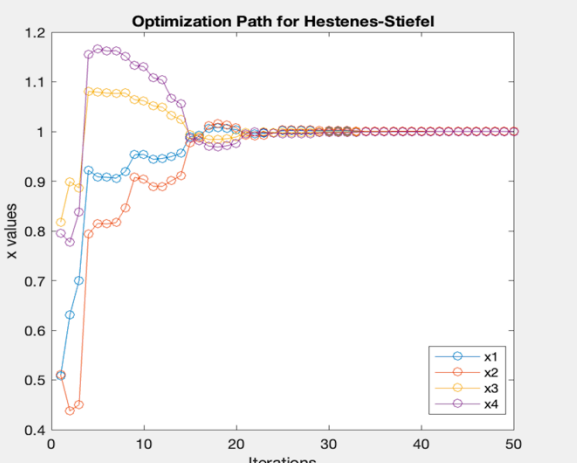

Fig. 2. Convergence path of each x values for Hestenes-Stiefel

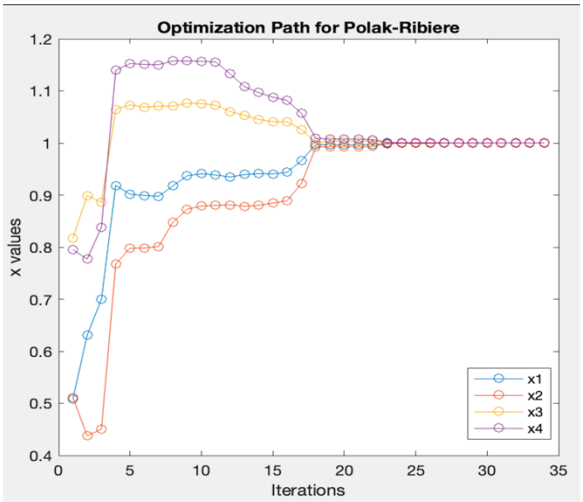

Fig. 3. Convergence path of each x values for Polak-Ribiere

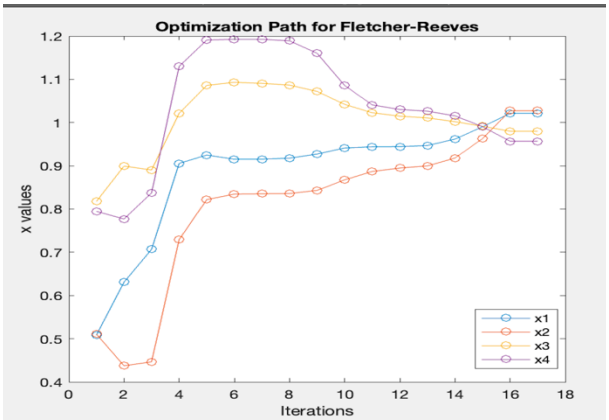

Fig. 4. Convergence path of each x values for Fletcher-Reeves

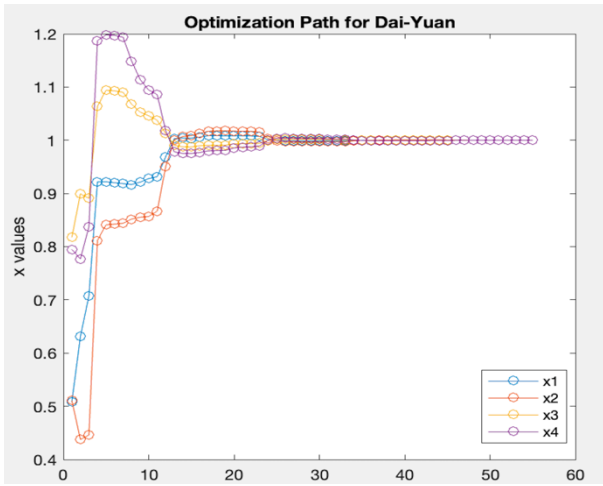

Fig. 5. Convergence path of each x values for Dai-Yuan

As we can see, we used the rand() function to assign an initial value and set our stopping criterion as the norm. Here, when our initial values start between 0 and 1, they can easily converge to our norm, which is our minimum. Fig. 1. we can see that Newton Raphson converged to 1 with very few iterations. If the function were quadratic, Newton would usually find the optimum in the first iteration.

Fig. 2., Fig. 3. and Fig. 5. their convergence is similar to each other. However, Hestenes and Dai-Yuan started to converge to 1 early after the 10th iteration. Polak Ribiere, on the other hand, only started on that path after the 15th iteration.

Other Hestenes-Stiefel and later algorithms converged in tiny tiny steps because they used conjugate gradients.. The Hestenes-Stiefel algorithm is one of the conjugate gradient methods and works efficiently especially for large dimensional problems. It quickly decreased the function value in a few steps.

The Polak-Ribiere algorithm is one of the conjugate gradient methods It decreased the function value in a short time and in a few iteration number.

The Fletcher-Reeves algorithm is one of the conjugate gradient methods. It reduced the function value steadily, although it did not take many steps. However, it was not very successful compared to others in converging to one while taking steps. Its performance was stable, but it did not approach the global minimum.

The Dai-Yuan algorithm is one of the conjugate gradient methods and can perform better especially for large size problems. It similar to the Hestenes-Stiefel algorithm quickly decreased the function value in a few steps.

2)If we change the stopping criteria as $|f(x_{k+1}) - f(x_k)| \leq \varepsilon$*:*

$$x_0 = [0.4324 \quad 0.8253 \quad 0.0835 \quad 0.1332]$$

II. PERFORMANCE

Algorithm	Execution Times	Iteration Number	Min value	Error
Newton-Raphson	1.011035	14	0.000000	0.000001
Hestenes-Stiefel	6.162950	49(NaN)	0.000000	0.000000
Polak-Ribiere	6.770772	55	0.000000	0.000000
Fletcher-Reeves	0.616023	7	1.049568	0.000000
Dai-Yuan	9.150773	75	0.000000	0.000000

As can be seen in Table II Newton-Raphson, Polak-Ribiere and Dai-Yuan algorithms are optimized algorithms since they give the results minimum. However, the Newton-Raphson algorithm has a shorter execution time, which makes it better than the others. At the same time, Fletcher-Reeves could not approach the global minimum despite the short execution time.

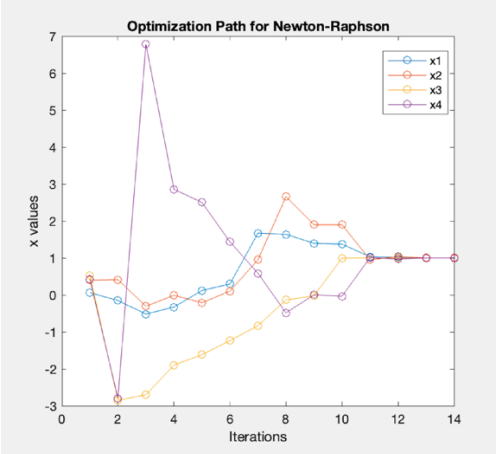

Fig. 6. Convergence path of each x values for Newton-Raphson

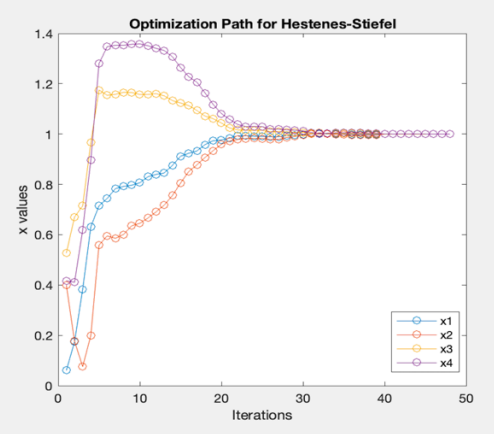

Fig. 7. Convergence path of each x values for Hestenes-Stiefel

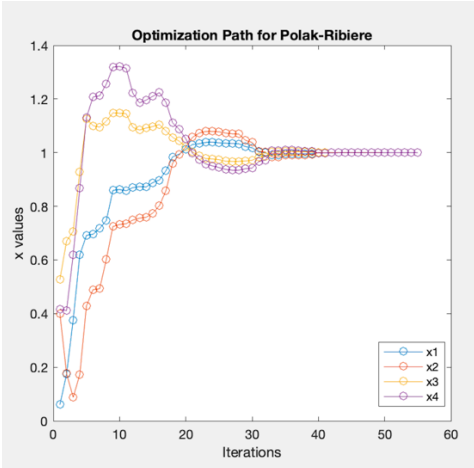

Fig. 8. Convergence path of each x values for Polak-Ribere

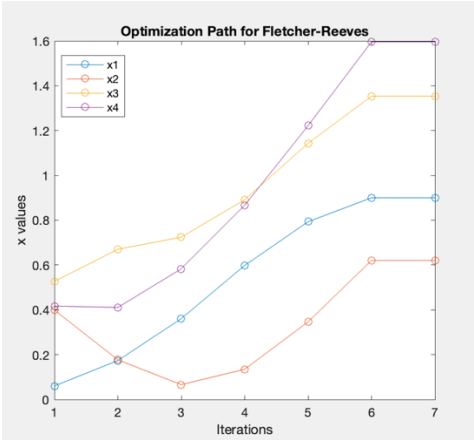

Fig. 9. Convergence path of each x values for Fletcher-Reeves

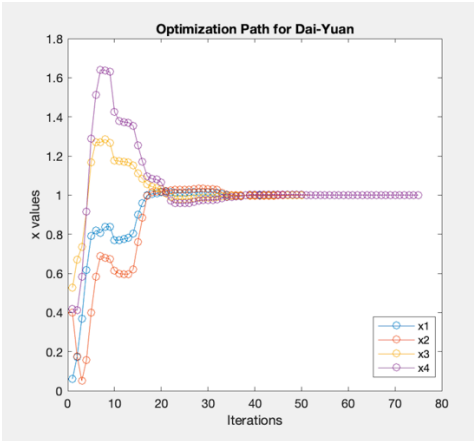

Fig. 10. Convergence path of each x values for Dai-Yuan

As we can see here, Newton is the algorithm that optimizes the initial point we set with rand() in absolute error. Although Feletcher-reeves shows fewer iterations and running time, it did not reach the global minimum.

Hestenes-Stiefel quickly decreased the function value in a few steps.

However, it failed in steps 48 and 49 as it produced NaN (Not a Number) values without finding an optimized solution. Although it reached the global minimum, there were problems with the stability of the algorithm.

The Polak-Ribiere algorithm reached the global minimum in a high step. The execution time is reasonable. The error is zero, which means it shows a stable approach.

The Feletcher-reeves algorithm shows a fast approach for these conditions and reaches the result in 7 steps. The execution time is quite short but the result is far from the global minimum. The error is the same compared to other algorithms. At the same time, when we look at the x_i* values table, it was not successful in converging to one.For this reason, it did not achieve the global minimum.

The Dai-Yuan algorithm achieves the result with the highest number of steps, so it has the longest execution time, but it achieves the global minimum. The absolute error is the same compared to the other conjugate gradient methods.

To summarize, the Newton-Raphson algorithm has reached the global minimum and has the fastest execution time. Dai-Yuan, Polak-Ribiere algorithms achieve the global minimum, although the computational cost is high.

A. Second Initial Guess:

Algorithms were tested for the second initial guess(randn()) as

$$x_0 = [-2.2584 \quad 2.2294 \quad 0.3376 \quad 1.0001]$$

1)Also, the stopping criteria is taken as $\|\nabla f(x_k)\| \leq \varepsilon$*:*

III. PERFORMANCE

Algorithm	Execution Times	Iteration Number	Min value	Error
Newton-Raphson	1.360448	13	0.000000	0.000055
Hestenes-Stiefel	8.939676	60	0.000000	0.000090
Polak-Ribiere	6.674286	46	0.000000	0.000069
Fletcher-Reeves	2.116712	23	0.000070	0.000017
Dai-Yuan	6.605421	46	0.000000	0.000053

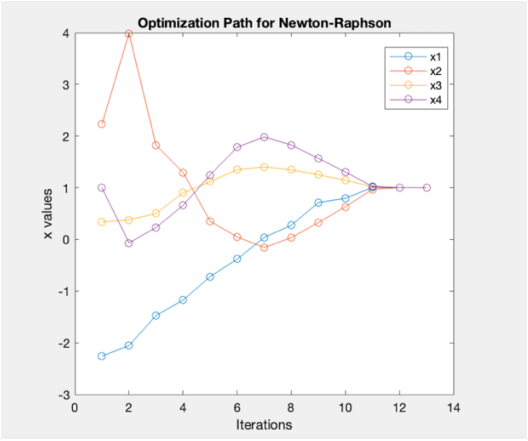

Fig. 11. Convergence path of each x values for Newton-Raphson

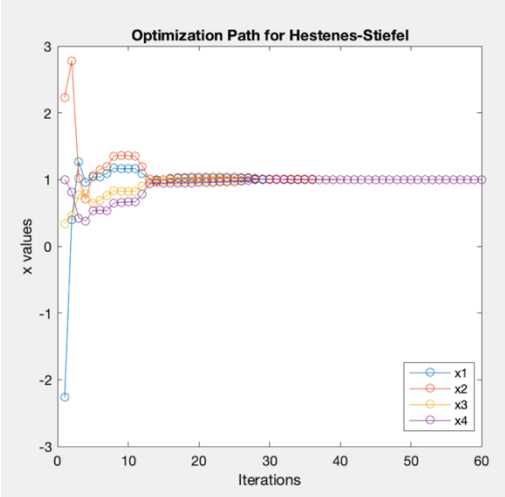

Fig. 12. Convergence path of each x values for Hestenes-Stiefel

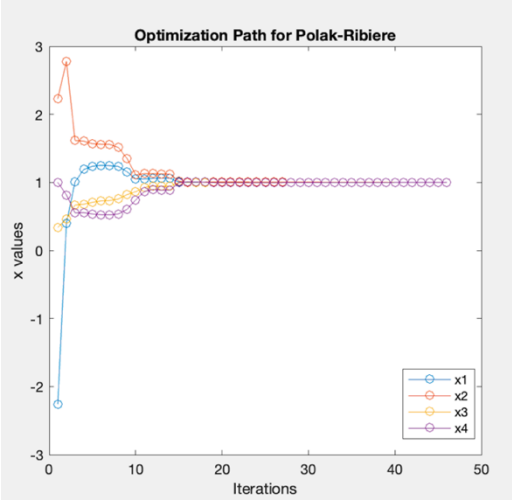

Fig. 13. Convergence path of each x values for Polak-Ribiere

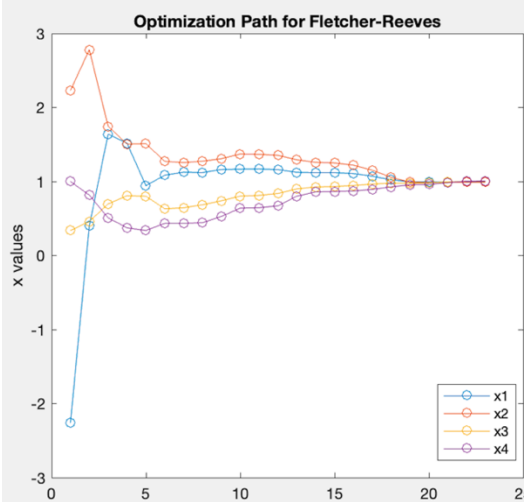

Fig. 14. Convergence path of each x values for Fletcher-Reeves

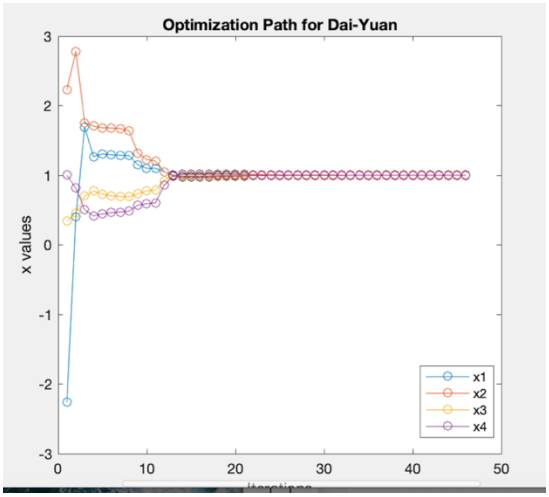

Fig. 15. Convergence path of each x values for Dai-Yuan

The Newton-Raphson Algorithm reached the correct solution the fastest and approached it with relatively low error.

In second place was the Fletcher-Reeves Algorithm. It reached the correct solution, but it took longer time and more steps than the Newton-Raphson algorithm, and its error rate is lower than the Newton-Raphson Algorithm.

Polak-Ribiere Algorithm and Dai-Yuan reached the correct result in the same number of iterations. However, Dai-Yuan is slightly faster and has a lower error rate than the other algorithm.

The Hestenes-Stiefel Algorithm has the longest number of steps and runtime. It also has the highest error rate.

2) If we change the stopping criteria as $|f(x_{k+1}) - f(x_k)| \leq \varepsilon$*:*

$$x_0 = [0.8261 \;\; 0.5362 \;\; 0.8979 - 0.1319]$$

IV. PERFORMANCE

Algorithm	Execution Times	Iteration Number	Min value	Error
Newton-Raphson	0.267128	5	0.000000	0.000013
Hestenes-Stiefel	2.062795	17(NaN)	0.001422	0.000000
Polak-Ribiere	4.912401	41	0.000000	0.000000
Fletcher-Reeves	1.620199	18	0.000820	0.000049
Dai-Yuan	4.864612	41	0.000000	0.000000

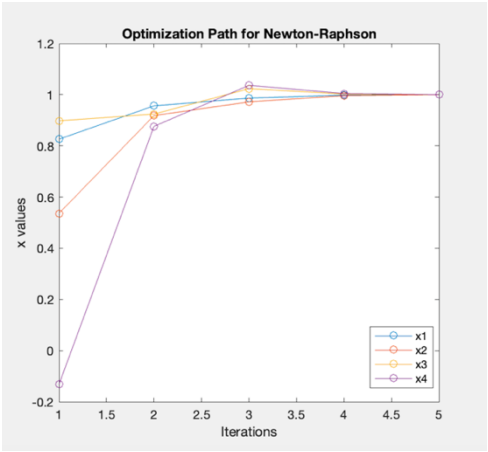

Fig. 16. Convergence path of each x values for Newton-Raphson

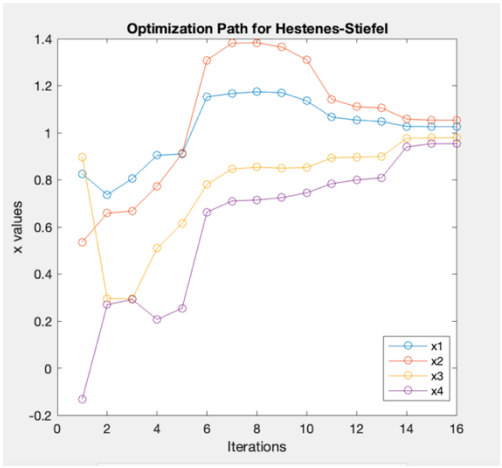

Fig. 17. Convergence path of each x values for Hestenes-Stiefel

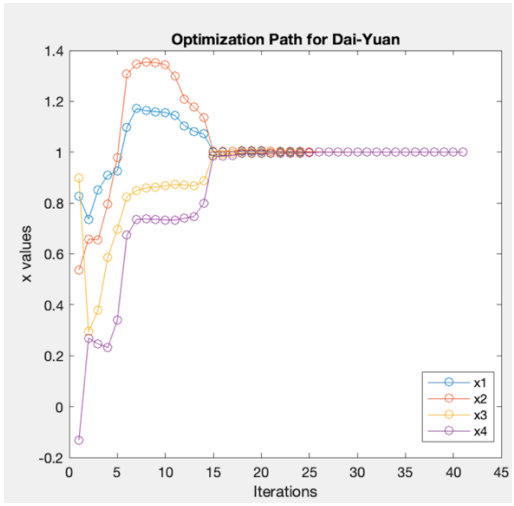

Fig. 18. Convergence path of each x values for Dai-Yuan

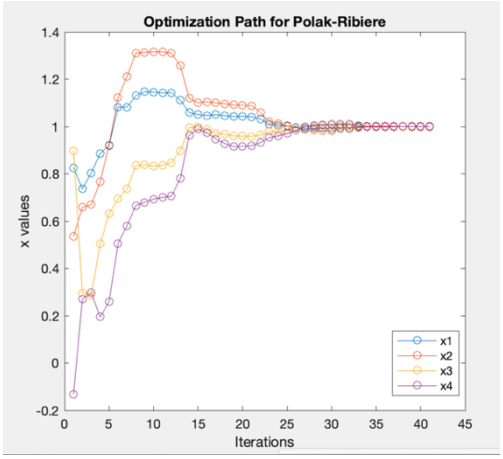

Fig. 19. Convergence path of each x values for Polak-Ribiere

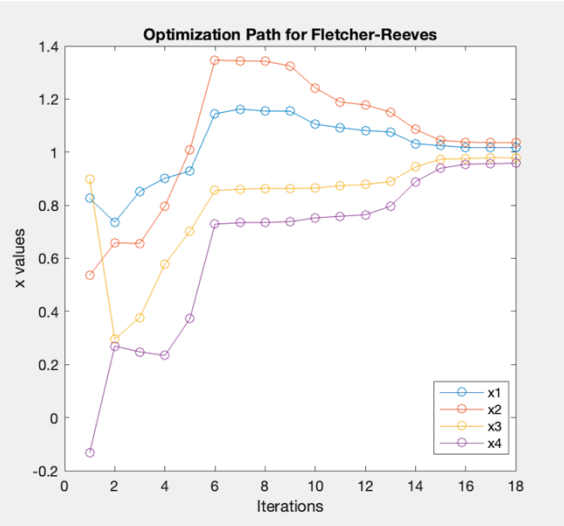

Fig. 20. Convergence path of each x values for Dai-Yuan

Again, like the others, the Newton-Raphson algorithm is the fastest algorithm to reach the global minimum.

The Hestenes-Stiefel algorithm did not reach the global minimum and failed due to the NaN error, although it is advantageous in terms of speed.

The Polak-Ribiere algorithm has reached the global minimum and has one of the lowest error rates.

The Fletcher-Reeves algorithm has a low number of iterations and a low time compared to other algorithms using conjugate gradients, but it does not achieve a result close to the global minimum.

The Dai-Yuan algorithm, similar to Polak-Ribiere, has the same number of iterations and runs in less time.

A. Third Initial Guess:

Algorithms were tested for the third initial guess([-2,2]) as

x_0 =[1.5460 -1.8853 -0.0404 -1.3283]

1) Also, the stopping criteria is taken as $\|\nabla f(x_k)\| \leq \varepsilon$*:*

V. PERFORMANCE

Algorithm	Execution Times	Iteration Number	Min value	Error
Newton-Raphson	1.024373	8	0.000000	0.000000
Hestenes-Stiefel	9.239306	64	0.000000	0.000019
Polak-Ribiere	12.147839	83	0.000000	0.000048
Fletcher-Reeves	2.272744	25	0.000258	0.000011
Dai-Yuan	1.309956	10(NaN)	-0.265373	5.522131

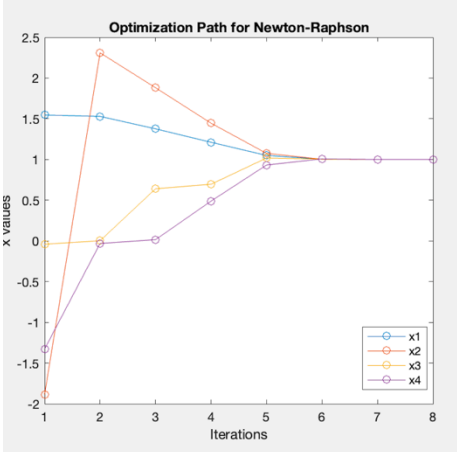

Fig. 21. Convergence path of each x values for Newton-Raphson

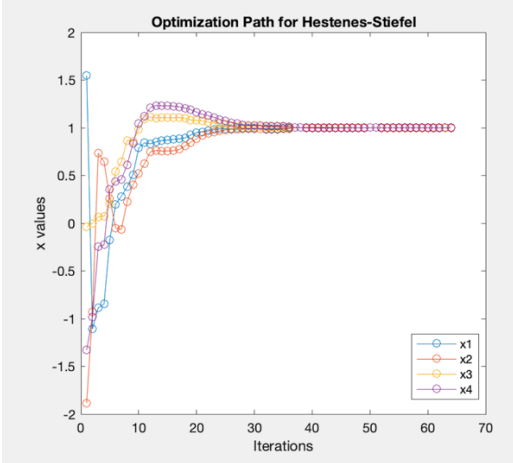

Fig. 22. Convergence path of each x values for Hestenes-Stiefel

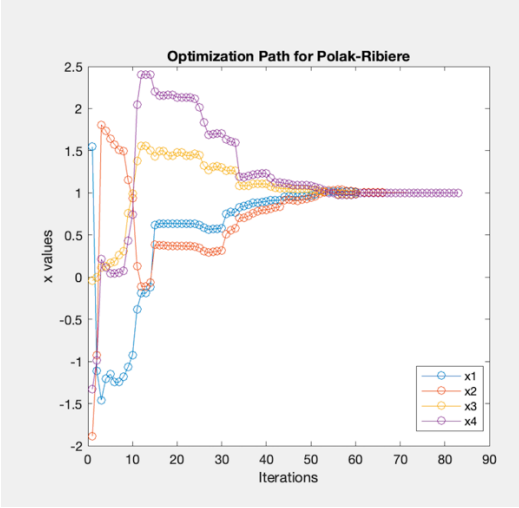

Fig. 23. Convergence path of each x values for Polak-Ribiere

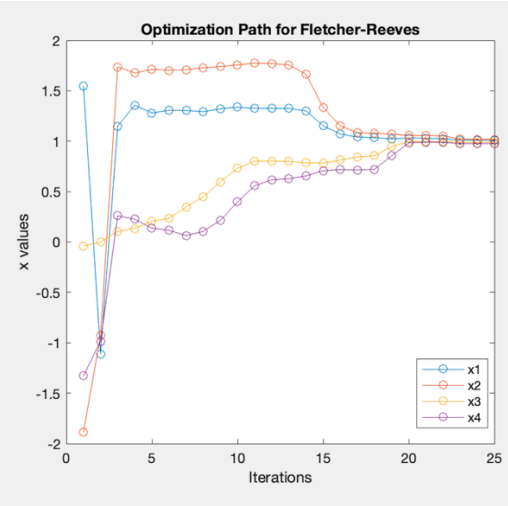

Fig. 24. Convergence path of each x values for Fletcher-Reeves

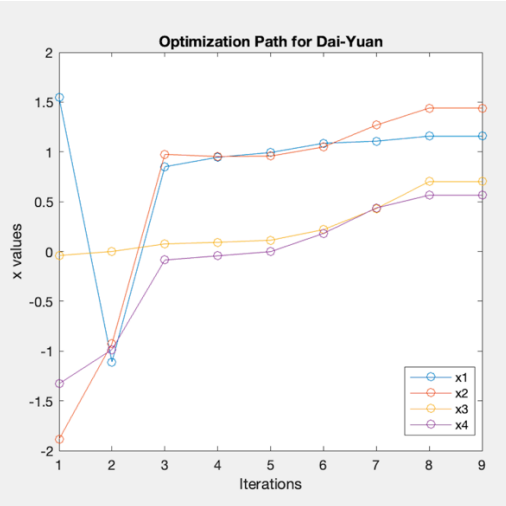

Fig. 25. Convergence path of each x values for Dai-Yuan

Here we have chosen our starting point to be the interval [-2,2]. Results based on this starting point;

The fastest and most accurate result is the Newton-Raphson algorithm, followed by the Dai-Yuan algorithm. However, although the number of iterations and runtime is low, it could not reach the global minimum and failed due to NaN error. Therefore, if we choose Feltcher-Reeves algorithm as the second algorithm in terms of speed and iterations, we would be wrong because it could not reach the global minimum. For this reason, Hestenes-Stiefel algorithm is chosen with less iteration, time and error, followed by Polak-Ribiere algorithm.

2)If we change the stopping criteria as $|f(x_{k+1}) - f(x_k)| \leq \varepsilon$*:*

$$x_0 = [0.6274 \quad 0.5119 \quad -0.8321 \quad -0.2734]$$

VI. PERFORMANCE

Algorithm	Execution Times	Iteration Number	Min value	Error
Newton-Raphson	1.061674	14	0.000000	0.000008
Hestenes-Stiefel	43.566581	57	0.000000	0.000000
Polak-Ribiere	34.072764	47	0.000000	0.000000
Fletcher-Reeves	18.852007	24	0.000202	0.000012
Dai-Yuan	27.360263	36	0.000000	0.000000

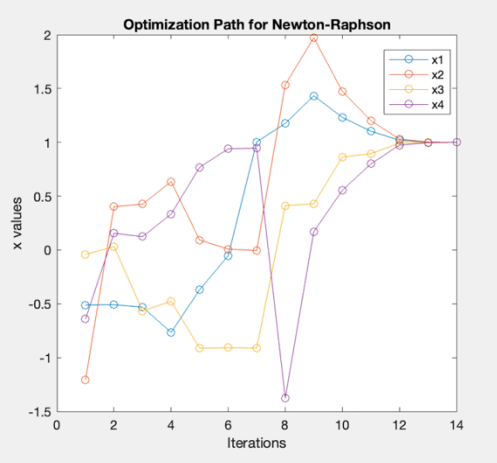

Fig. 26. Convergence path of each x values for Newton-Raphson

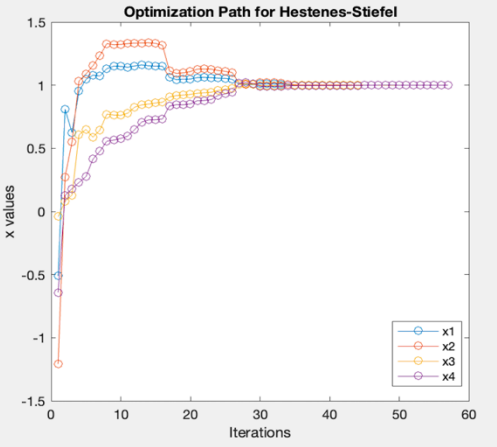

Fig. 27. Convergence path of each x values for Hestenes-Stiefel

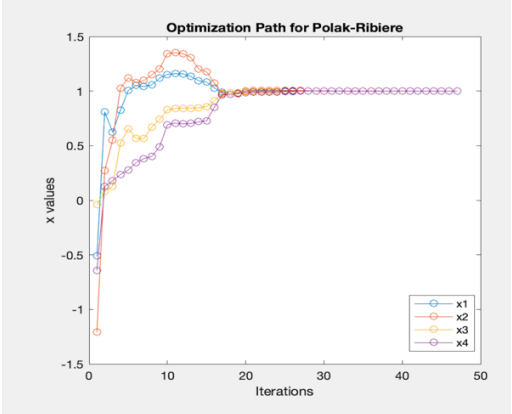

Fig. 28. Convergence path of each x values for Polak-Ribiere

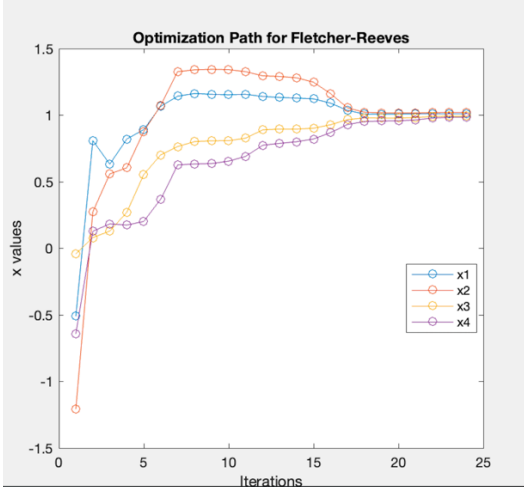

Fig. 29. Convergence path of each x values for Fletcher-Reeves

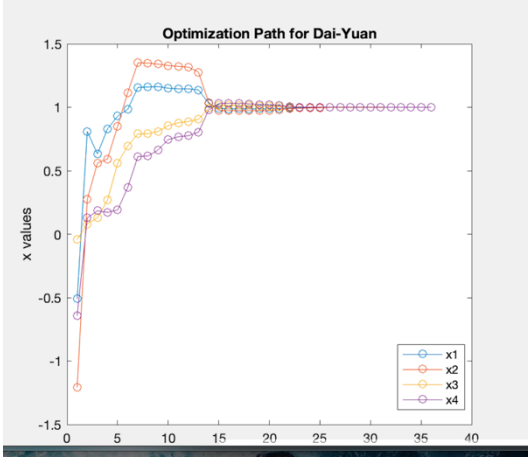

Fig. 30. Convergence path of each x values for Dai-Yuan

Here, when calculating the result with absolute error, there were too many nan values, so I set the alpha value to 100000. For this reason, the algorithm run times are much higher than other results.

The fastest and most accurate result is the Newton-Raphson algorithm, followed by the Fetcher-Reeves algorithm. However, although the number of iterations and running time were low, it could not reach the global minimum. So the second algorithm, Dai-Yuan, is better than the others in terms of speed and iterations. It is followed by Polak-Ribiere. Then comes Hestenes-Stiefel.

If we look at the convergence graphs, we can say that Dai-Yuan, Polak-Ribiere and Hestenes-Stiefel gave similar results.

V. CONCLUSION

The second stopping criterion optimizes the performance of the algorithms and provides stable and accurate results in fewer steps and in less time. This shows how important the stopping criterion is in optimization problems and that the choice of an appropriate criterion for the function directly affects the success of the algorithm.

The choice of stopping criterion and error bound plays a critical role in the performance of optimization algorithms. Smaller criteria improve accuracy but also increase computational cost, while larger criteria speed up the process but reduce accuracy.

Approximations depend on the initial condition. This is a general property for all optimization algorithms. A well-chosen starting point allows the algorithm to approximate quickly and accurately. However, a poorly chosen starting point may cause the algorithm not to converge or to produce slow and inaccurate results. When we look at the best results, we see that using rand(), the best results converge to one faster and have less running time.

In terms of overall performance analysis, the Newton-Raphson algorithm is the most reasonable choice.

REFERENCES

[1] Laskari, E. C., Parsopoulos, K. E., & Vrahatis, M. N. (2003). Evolutionary operators in global optimization with dynamic search trajectories. *Numerical Algorithms*, *34*, 393-403.

[2] Beiranvand, V., Hare, W., & Lucet, Y. (2017). Best practices for comparing optimization algorithms. Optimization and Engineering, 18, 815-848.

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper