Assignment 3 Report

AEROSP 588 – Dr. Joaquim Martins

University of Michigan

October 7, 2025

Austin Leo Thomas

I. Problem 3.2

The final algorithm selected is a conjugate gradient approach utilizing a line search satisfying the strong Wolfe conditions. The Python optimization script authored for this assignment also allows for the implementation of a steepest descent approach as well as line search via backtracking; however, as will be shown later, the best performance was achieved (expectedly) with a conjugate gradient approach and a line search method based on the strong Wolfe conditions.

The β-value employed to achieve conjugacy in the search direction determination algorithm was found via the Fletcher-Reeves formula [1]. This formula was selected over the Polak-Ribière formula to: (a) enforce a more stringent μ_2-value which is less than 0.5; (b) prevent the addition of an enforcement criteria that β remain non-negative; and (c) allow for increased control over when the search direction vector reverts back to the direction of steepest descent.

The selected search direction reset criteria was initially planned to be the heuristic approach of resetting to the direction of steepest descent every n iterations, where n is the dimensionality of the problem. However, in implementing this approach is quickly became clear that for two-dimensional problems such a criteria resulted in an algorithm only marginally different from the steepest descent approach: resetting every two iterations led to every other search direction simply being the direction of steepest descent. As such, the "more mathematical" reset condition, defined by Eq. (1), was chosen. In practice, it was found that this led to search direction resets occurring every three to four iterations for a simple problem such as the slanted quadratic function (discussed later).

$$\frac{\left|\nabla f_k^{\mathrm{T}} \nabla f_{k-1}\right|}{\left|\nabla f_k^{\mathrm{T}} \nabla f_k\right|} \geq 0.1 \qquad (1)$$

Another important detail of the algorithm was the choice of interpolation method for the pinpointing step of the line search algorithm. Since minimizing function evaluations was desirable above all else, a simple bisection formula, shown in Eq. (2), was chosen to determine the α_p-value for each pinpointing attempt. This prevented any additional evaluations of $\phi(\alpha)$ and $\phi'(\alpha)$; though this approach may result in more pinpointing loops per single iteration, it means there is one less function evaluation per pinpointing loop, which in theory proves profitable given the presented premise.

$$\alpha_p = \frac{\alpha_{high} + \alpha_{low}}{2} \qquad (2)$$

As is required for most optimization algorithms, hard-coded kill switches were included on all algorithm elements which employed while-loops. Namely, the bracketing and pinpointing loops were limited to 100

iterations, and the full optimization loop was limited to 2000 iterations.

Quasi-Newton methods, such as a BFGS approach, were intentionally avoided for heuristic reasons. Since future assignments will make use of and allow for analysis of such methods, it was decided to restrict the scope of this evaluation to non-Newton or quasi-Newton approaches. As such, formulation of this algorithm considered only combinations of two line search methods – backtracking and strong Wolfe line search – and two search direction determination methods – steepest descent and conjugate gradient approaches. Of these, a conjugate gradient approach with a line search based in the strong Wolfe conditions was the best.

II. Problem 3.3

Two two-dimensional test problems were applied to the completed algorithms to ascertain which combination of line search method and search direction determination method was the best. These problems were the minimization of the slanted quadratic function, defined in Eq. (3), and the two-dimensional Rosenbrock function, defined in Eq. (4) [1].

$$f(x) = x_1^2 + x_2^2 - \beta x_1 x_2 \quad (3)$$

$$f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 \quad (4)$$

These functions were optimized with five different algorithms, defined in Table (I). The parameters passed to the algorithms for this investigation are defined in Table (II). The results of the optimization processes, including the number of function evaluations required to reach the optimal (or non-optimal) point, are defined in Table (III) for the slanted quadratic function and Table (IV) for the two-dimensional Rosenbrock function. Those algorithms which did not converge to the analytic optimum, and were stopped at the 2000-iteration limit, are marked with an asterisk. Initial guesses of $x_0(1,2)$ and $x_0(0,0)$ were used for the slanted quadratic function and two-dimensional Rosenbrock function, respectively.

Table I
Algorithm Definitions

Alg.	Search Direction	Line Search
0	SciPy CG Optimizer Function	
1	Conjugate Gradient	Strong Wolfe
2	Steepest Descent	Strong Wolfe
3	Conjugate Gradient	Backtracking
4	Steepest Descent	Backtracking

Table II
Algorithm Parameters

Parameter	Value
α_{init}, initial step size	1
μ_1, sufficient decrease factor	$1 \cdot 10^{-3}$
μ_2, sufficient curvature factor	0.4
σ, step size increase factor	2
ρ, backtracking factor	0.5
τ, convergence tolerance	$1 \cdot 10^{-6}$

Table III
Slanted Quadratic Function Solutions

Alg.	x^*	$f(x^*)$	Function Calls
0	(0,0)	0	7
1	(0,0)	0	743
2	(0,0)	0	988
3	(0,0)	0	457
4	(0,0)	0	366

Table IV
2D Rosenbrock Function Solutions

Alg.	x^*	$f(x^*)$	Function Calls
0	(1,1)	0	42
1	(1,1)	0	5,771
2	(1,1)	0	45,576
3*	(0.97,0.94)	0.001	26,703
4*	(0.97,0.94)	0.001	25,413

While much of the motivation for the selected algorithm described in *Section I* was of a heuristic and theoretical nature, this investigation has provided sound evidence that, of the four potential algorithms, the conjugate gradient approach with a line search based on the strong Wolfe conditions is the best approach.

Examination of Table (III) might lead one to believe this is not the case: all four methods converged to the solution, with the simplest algorithm, Algorithm 4, doing so with the fewest number of function calls. Indeed, from this is may be reasonable to conclude that, for simple functions such as the slanted quadratic, a simple optimization approach – in this case, the steepest descent method with backtracking – is best.

However, the goal of this exercise was not to develop an algorithm which is extremely efficient at solving only simple problems but instead to develop an algorithm that can prove relatively efficient in solving even complex problems. Examination of Table (IV) proves that Algorithm 1 is such a choice. Firstly, it is seen that Algorithms 3 and 4, which utilize a backtracking line search, do not converge to the correct solution, even after 2000 iterations and over 20,000 function calls. This is a disappointing result and eliminates backtracking-based methods from serious consideration for the development of a robust optimizer. Algorithms 1 and 2, both enforcing the strong Wolfe conditions for line search but taking a different approach for search direction determination, both converge to the correct optimum. However, Algorithm 1, which employs the theoretically (and clearly experimentally) superior conjugate gradient approach to search direction determination, converges to the optimum while using almost one tenth of the function calls Algorithm 2 requires.

Thus, based on these experimental results with both a simple and complex optimization problem, it is clear that Algorithm 1 is the only method which is both robust and computationally inexpensive at scale.

It should briefly be noted that, compared to the SciPy CG optimizer (Algorithm 0), none of these approaches hold up. The SciPy optimizer is, indeed, an 'optimized optimizer': it accomplishes its goal while remaining nearly and computationally inexpensive as possible. The goal of this investigation was not to 'beat' this algorithm – doing so would be very difficult indeed. Though it is useful to see how such an ideal optimizer compares to the more rudimentary solutions presented by Algorithms 1 through 4, the primary purpose of this investigation was to prove, experimentally, that which was posited in *Section I*: that of the non-Newton or quasi-Newton based nonlinear, unconstrained optimizer combinations, a conjugate gradient approach with strong Wolfe line search is the most robust and computationally efficient.

III. Problem 3.4

From here, additional analysis was undertaken for the solutions to the slanted

quadratic and two-dimensional Rosenbrock functions found by Algorithm 1 (henceforth referred to as 'the selected algorithm'). One interesting metric of an optimizer is a convergence plot, which has as its abscissa the number of iterations in the solution and as its ordinate the value of the maximum component of the gradient vector, $\|\nabla f\|_\infty$. It is this value which is compared to the convergence tolerance, τ, upon each iteration of the optimizer, and thus it is this value which ultimately determines the efficiency of the optimizer: the faster this value can be driven below τ, the more efficient the algorithm. The convergence plots for the slanted quadratic and two-dimensional Rosenbrock functions found by the selected algorithm are shown in Fig. (1) and (2), respectively.

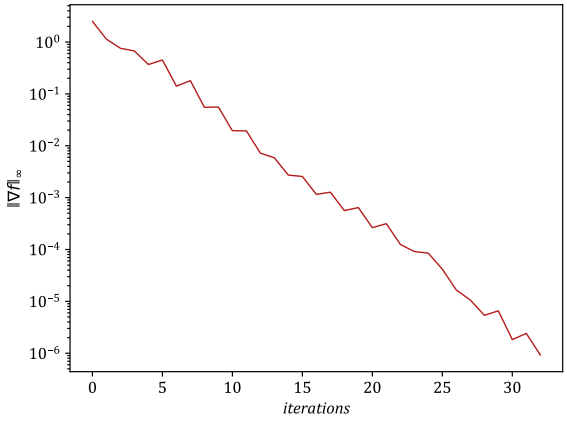

Figure 1. Convergence plot for the selected algorithm's solution of the slanted quadratic function, with the number of iterations (not function calls!) indicated as the abscissa and the maximum component of the gradient vector indicated as the ordinate (note the logarithmic scale on the ordinate axis).

Examination of Fig. (1) reveals very orderly convergence behavior when the selected algorithm is applied to a simple func-

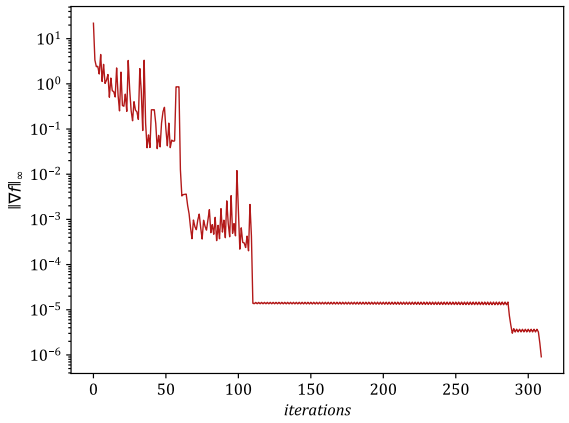

Figure 2. Convergence plot for the selected algorithm's solution of the two-dimensional Rosenbrock function, with the number of iterations (not function calls!) indicated as the abscissa and the maximum component of the gradient vector indicated as the ordinate (note the logarithmic scale on the ordinate axis).

-tion such as the slanted quadratic. The convergence plot is linearized when plotted with a logarithmic scale on the ordinate axis, revealing a order-of-magnitude decrease in $\|\nabla f\|_\infty$ for every five iterations of the selected algorithm. Such orderly behavior is indicative of a well-behaved function which the selected algorithm has no issue optimizing, given sufficient time: the progress in optimization is steady, and tuning the parameters defined in Table (II), such as increasing the initial step size or the step size increase factor, would likely allow for faster convergence.

Examination of Fig. (2) reveals the opposite behavior in the solution of the two-dimensional Rosenbrock function. The convergence plot is highly disorderly, with significant noise and considerable, unpredictable variations in $\|\nabla f\|_\infty$ from iteration to iteration. What is even more curious than the noisy sections of this

convergence plot are the static portions: a long, nearly flat segment appears between Iterations 105 and 290 (approximately) before a significant decrease in $\|\nabla f\|_\infty$. This is followed by another flat segment from Iterations 290 to 305 before another jump sets $\|\nabla f\|_\infty$ just below τ.

Such behavior is strange indeed, but not entirely out of the ordinary for conjugate gradient approaches; indeed, Fig. (4.58) in *Engineering Design Optimization* includes a conjugant gradient convergent plot which shows similar behavior: long, nearly flat segments sandwiches between highly erratic and noisy regions. Such behavior is especially expected for a function as difficult and ill-conditioned as the Rosenbrock function.

Together, these convergence plots allow for very useful insight into the selected algorithm. Though it was seen earlier that the algorithm is robust – able to converge relatively quickly when faced even with difficult optimization problems – the algorithm still clearly suffers under such problems. The algorithm arrives at a solution under such conditions, but certainly much less efficiently than another, say quasi-Newton algorithm might. For simpler problems, however, it is clear that the selected algorithm is more than suitable, especially if one were to tune the parameters described in Table (II).

IV. Problem 3.5

Optimization path plots were also generated for the solutions found by the selected algorithm. These plots trace the path the optimizer takes through the design space from iteration-to-iteration and are overlayed on a contour plot of the function being optimized. Such plots are useful for the visualization of an algorithm's efficiency: a more direct path correlates to a more efficient algorithm. The path plots for the slanted quadratic function and two-dimensional Rosenbrock function solutions are shown in Fig. (3) and (4), respectively.

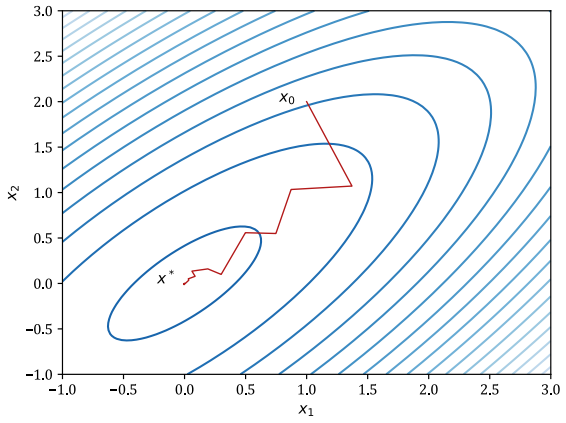

Figure 3. Optimization path for the slanted quadratic function solution found by the selected algorithm. The starting point, $x_0(1,2)$, and optimum point, $x^*(0,0)$, are marked. Darker contour lines indicate smaller f-values.

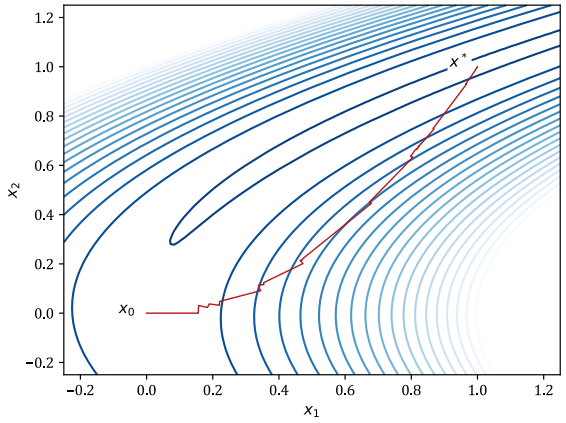

Figure 4. Optimization path for the two-dimensional Rosenbrock function solution found by the selected algorithm. The starting point, $x_0(0,0)$, and optimum point, $x^*(1,1)$, are marked. Darker contour lines indicate smaller f-values.

These optimization path plots provide wonderful insight into some of the behavior seen in the convergence plots shown in *Section III* as well as some potential shortcomings of the selected algorithm.

Examination of Fig. (3) reveals that the optimization path is heavily zig-zagged, a characteristic of the steepest descent approach which is clearly still present in this algorithm. This behavior could be due to a variety of reasons. It is possible that the search direction is resetting to steepest descent too often; if this is the case, solving for β via the Polak-Ribière formula and resetting when $\beta \to 0$, rather than applying the Fletcher-Reeves formula and resetting when Eq. (1) is satisfied, might prove beneficial [1]. Alternatively, this behavior might be the result of a too-large initial step size, α_{init}, an insufficient sufficient decrease factor, μ_1, or a too-large sufficient curvature factor, μ_2. Any of these inadequacies could cause the line search to overshoot the 'best' point in the search direction, leading to zig-zagging behavior.

Examination of Fig. (4) indicates that the opposite may be true for the two-dimensional Rosenbrock solution: whereas the steps seemed to be too large in the slanted quadratic solution, here they seem to be too small. While the optimization path does occasionally take significant steps in the right direction, the iterations generally seem to be quite close to one another, indicating a failure of some level in the line search procedure: while the optimizer is pointed in the right direction, the steps between iterations are too small. This could again be remedied by tuning of the optimizer parameters: namely, increasing the initial step size, α_{init}, or increasing the step size increase factor, σ, would encourage the algorithm to branch further in the search direction with each iteration. However, given the difficulty associated with the two-dimensional Rosenbrock function, such tuning would be difficult.

It should be noted that these parameters were purposefully not tuned for this investigation. As part of the exploration process, parameters were left static and set at baseline recommended values found in *Engineering Design Optimization* [1]. This allowed for a critical look at the selected algorithm in a flawed state, requiring analysis of the optimizer's behavior to deduce possible improvements. It also allowed for an unbiased comparison of the different algorithms presented in *Section II*: since none of the algorithms were tuned, and all had the same parameters, the comparison was fair.

V. Problem 3.6

As previously stated, the goal of this investigation was to create a robust and computationally efficient algorithm. In this spirit, a sort of endurance test was applied to the selected algorithm: it was presented with increasingly high-dimensional Rosenbrock problems, and the number of iterations required for the algorithm to converge was recorded. The goal of this exercise was to investigate the relation between the number of iterations required for convergence and the dimensionality of the function being optimized. Linear behavior would be good here, whereas exponential behavior would indicate a serious issue in problem scaling.

Each time the algorithm successfully optimized an n-dimensional Rosenbrock function, it was subsequently presented with a $2n$-dimensional Rosenbrock function to optimize, beginning with $n = 2$. The selected algorithm was successfully able to optimize

the functions up to $n = 64$. When attempting $n = 128$, the algorithm reach its 2000-iteration limit. Though theoretically higher-dimensional problems would be attempted by increasing the iteration restriction on the algorithm, a sample size up to $n = 64$ ought to be sufficient for this investigation. The resultant plot of required iterations versus Rosenbrock function dimensionality is shown in Fig. (5). Note that for each solution the initial guess point was the origin of the n-dimensional design space. Additionally, due to the complexity of the Rosenbrock function, the convergence tolerance was loosened for this investigation from $\tau = 1 \cdot 10^6$ to $\tau = 1 \cdot 10^3$.

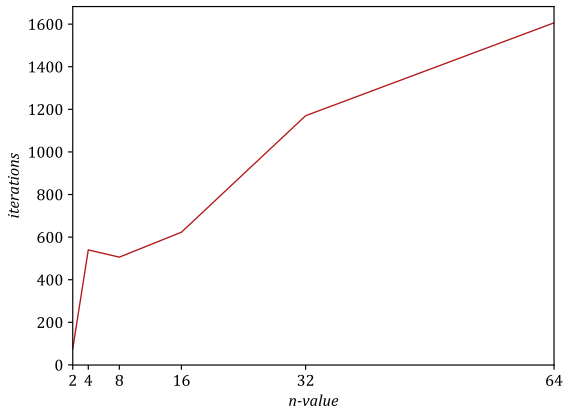

Figure 5. Plot reflecting the computational cost of the selected algorithm with increasingly high-dimensional Rosenbrock functions. The abscissa reflects the dimensionality of the function to be optimized while the ordinate reflects the number of iterations required for the selected algorithm to reach convergence. Note the lack of logarithmic scaling on either axis.

Examination of Fig. (5) reveals that, remarkably, the relationship between required iterations and problem dimensionality is relatively linear: more sample sizes may be needed to bolster this claim, but no clear exponential relationship is evidenced for $n \leq 64$. If this computational cost relationship is truly non-exponential (even if it is not perfectly linear), this makes the selected algorithm very robust indeed, allowing for near limitless problem scaling without skyrocketing computational costs. Such behavior is indeed highly desirable in optimization algorithms.

VI. Problem 3.7

This assignment was an excellent learning opportunity both into the implementation of steepest descent and conjugate gradient approaches to nonlinear unconstrained optimization as well as to the limitations and benefits of these methods.

As with the previous assignments, the main challenges centered around coding and debugging of the algorithms. The Python scripts written for this assignment were relatively lengthy and took some time to fully work out. I additionally found that I had some critical misunderstandings of the conjugate gradient approach following lecture and note-taking from the textbook. I had to thoroughly re-read the mathematical derivations involved to truly understand what was going on, especially regarding the Flecther-Reeves formula and the motivation behind it.

For this reason I am very glad that I chose to focus this assignment specifically on the conjugate gradient method. Had I focused instead on quasi-Newton methods, I likely would have been left with a poor understanding of conjugate gradient approaches leaving this course.

Regarding areas of improvement, I believe I have covered most in some detail

throughout this report. The main point of improvement which could be made to the selected algorithm is, of course, parameter tuning. When implementing this algorithm in the future I will certainly tune parameters, though I am glad that I did not here, as it allowed for some meaningful thought and analysis into the shortcomings of the algorithm.

Overall, this was a very useful and insightful investigation into the conjugate gradient approach, and I am happy with the algorithm which I produced, even if it will shortly be replaced with a quasi-Newton approach in future assignments.

VII. Disclosures

As required by the syllabus, a disclosure is included here regarding AI use on this assignment. U-M GPT was used as an aide when writing the Python scripts employed in this assignment. The tool was used primarily in place of a search engine to look up syntax for the various modules used in the scripts. The tool was also used in the debugging process when other measures failed. AI did not write any blocks of code submitted for this assignment, nor was the tool used in any part of the authoring of this report.

VIII. References

[1] J. R. Martins and A. Ning, Engineering Design Optimization, Cambridge: Cambridge University Press, 2020.