A comparison of trust region methods in unconstrained optimization

Ritik Kothari
Department of Astronomy and Astrophysics, University of Toronto

(Dated: May 5, 2023)

Abstract

In this paper, we analyze the numerical optimization strategy that makes use of trust-region methods in the context of non-linear and non-convex problems. In particular, we consider the effectiveness of the dogleg method in solving the unconstrained trust region subproblem. To address the limitations that trust-region methods may encounter with non-convex problems, we examine the Retrospective Trust Region Method (RTRM), which uses the current iterate instead of the previous one to make approximations. We then compare the efficacy of the RTRM algorithm with the dogleg method for a test suite of optimization benchmark functions. The performance of each algorithm is measured through the accuracy and number of iterations taken throughout the optimization process, as well as their convergence rates. It is found that both the dogleg method and RTRM are highly accurate in their optimization, yet vary significantly in rapidness. We conclude in agreement with Bastin et al. (2007) concerning the competitiveness of the retrospective trust-region method.

I. INTRODUCTION

Numerical optimization is a important area of research that has considerable influences on many scientific, economic, and mathematical fields. Typical applications of optimization in these contexts involve minimizing or maximizing objective functions, which are the optimization's goals that can be subject to specific constraints. In general, many optimization problems take the form

$$\min_{x \in X} f(x)\,, \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a twice differentiable objective function and $X \subseteq \mathbb{R}^n$ is the set of feasible solutions [1, 5]. This type of problem can be solved using line search algorithms, which iteratively find a linear path to the optimal solution of an objective function, or by trust-region methods, which attempt to optimize an approximation of the objective function by iterating over neighbourhoods of optimal points. For the focus of this paper, we consider the efficacy and strength of trust-region methods, which seek to solve the subproblem

$$\min_{d \in X_k} m_k(d) \text{ s.t. } \|d\|_{W_k} \leq \Delta_k \tag{2}$$

where $m_k(d)$ is a model function that approximates $f(x_k + d)$ near the current iterate point x_k, X_k is an approximation of the shifted feasible set $X - x_k$, $\|.\|_{W_k}$ is a norm in $\mathbb{R}^n$, and $\Delta_k > 0$ is the radius of the trust region [5]. Of the various optimization strategies commonly used today, trust-region methods have gained popularity as a powerful way of addressing non-linear and non-convex problems.

Trust-region algorithms are a class of optimization techniques that operate by creating an approximation m_k of an objective function f within the neighbourhood of the algorithm's current estimate of the optimal solution. This estimate is known as the current iterate point x_k. The size of the neighbourhood's radius Δ_k determines where the algorithm decides m_k to be a reasonable approximation of the objective function. If the algorithm finds that m_k is a poor approximation of the objective function at the current iterate x_k, the radius Δ_k is made smaller and a new model is found [3].

As a result, the size of the trust region is imperative to the success of each step [3]. Smaller-than-required values of Δ_k imply that the method is prevented from taking a significant step towards the minimizer of the objective function, whereas values too large pose the risk of the model's minimizer being too far away from the minimizer of the objective function [3]. Generally, the step's direction is altered depending on how the size of the trust region changes [3]. In practical settings, the size of the region is chosen in line with the algorithm's performance during previous iterations. A consistently reliable model allows for more ambitious increases in the trust region's radius, shortening the time it takes to reach the objective function's optimal solution [3].

A particular technique for solving the trust-region subproblem is the dogleg method, which can be used when the Hessian matrix $B = \nabla^2 f(x)$ is positive definite [3]. The dogleg method employs ideas from the Newton and steepest descent directions, and combines their benefits [2].

While traditional trust-region methods, such as the dogleg method, are adept at optimizing a variety of situations, there are certain limitations that affect their performance. In particular, Nocedal and Wright (2006) conclude that the dogleg method is typically most appropriate for convex optimization problems, implying that it is less suited for non-convex problems [3]. Furthermore, the dogleg method's dependence on a local quadratic model to create approximations raises the potential of it being ineffective in optimizing non-convex problems.

To try and address these limitations, Bastin et al. (2007) propose a trust-region optimization method that considers the current iterate, instead of the previous one, in creating its approximation [1]. This Retrospective Trust Region Method (RTRM) addresses the aforementioned issues by taking into account a greater number of trust-region size adjustments, possibly leading to more accurate convergence [1].

In this paper, we introduce the RTRM algorithm created by Bastin et al. (2007), and compare its efficacy to that of the dogleg method for unconstrained optimization problems. We begin with a brief review of the mathematical theory involved in trust-region methods, and expand to also introduce the theory of the RTRM in Section II. We then consider the strengths and weaknesses of each method by evaluating their performance through several numerical experiments in Section III. Results and findings are discussed in Section IV, and conclusions are made in Section V.

II. THE MATHEMATICAL THEORY OF TRUST-REGION METHODS

In the previous section, we introduced trust-region methods as a class of iterative optimization algorithms that optimize non-linear and continuous objective functions. Specifically, trust-region methods are well-suited for problems of several variables. In this section, we discuss the mathematical theory behind trust-region methods, and provide descriptions of the RTRM and dogleg method.

A. Trust-Region Method Fundamentals

In this section, we provide an overview of the fundamental mathematical theory behind trust-region methods, using [3] as a reference.

To begin, note that the approximation m_k of the objective function f is usually quadratic in nature when considering trust-region optimization [3]. It is based on the Taylor expansion of $f(x_k)$ around the iterate point x_k, given by

$$f(x_k + d) = f_k + g_k^T d + \frac{1}{2} d^T \nabla^2 f(x_k + td)\, d\,, \tag{3}$$

where $f_k = f(x_k)$, $g_k = \nabla f(x_k)$, and $t \in (0, 1)$ is some scalar [3]. Taking a symmetric matrix B_k to be the Hessian approximation, we then obtain

$$\min_{d \in X_k} m_k(d) = f_k + g_k^T d + \frac{1}{2} d^T B_k d \qquad \text{s.t.} \quad \|d\|_{W_k} \le \Delta_k\,. \tag{4}$$

Note that Equation (3) resembles the subproblem given by Equation (2), in its expanded form. In the case where $B_k = \nabla^2 f(x_k)$, where $\nabla^2 f(x_k)$ is the Hessian matrix of f, the trust-region method becomes a trust-region Newton method [3].

Figure 1 depicts the trust-region approach for an objective function $f : \mathbb{R}^2 \to \mathbb{R}$, in which the current iterate x_k and the function's minimizer x^* lie at opposite ends of a curved valley [3]:

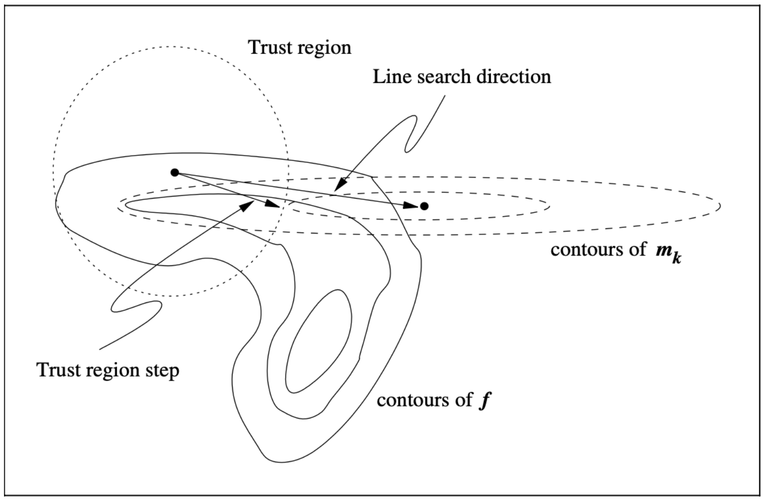

FIG. 1: Figure 4.1 from [3]. The quadratic model function m_k, shown by the dashed ellipses, is constructed from information about f and f' at x_k and potentially also by previously obtained information. A line search method based on this model searches along the step to the minimizer of m_k, however its direction will result in a small reduction in f, at best. The trust-region method steps to the minimizer of m_k within the dotted circle (shown), which yields a more significant reduction in f and hence shows better progress toward the solution.

The key idea of the trust region approach is to solve the subproblem described by Equation (4) iteratively, where the objective function and its constraints are quadratic [3].

Now, trust-region methods only require approximate solutions in order to achieve convergence. This starts with first choosing a reasonable value for the trust-region radius Δ_k, which is based on reconciling f and m_k from previous iterations of the algorithm [3]. For a step d, the ratio ρ_k is defined as

$$\rho_k = \frac{f(x_k) - f(x_k + d)}{m_k(0) - m_k(d)}\,, \tag{5}$$

where the numerator is the reduction in f, and the denominator is the predicted reduction by the model function m_k [3]. Note that since m_k is minimized over a region which includes $d = 0$, we require that $m_k(0) - m_k(d)$ be nonnegative [3]. The importance of Equation (5) lies in how it is used to determine the radius of the trust-region algorithm's next iteration. If $\rho_k < 0$, then $f(x_k + d) > f(x_k)$, implying that the step d is a poor choice. This leads to a recalculation being required for the value of Δ_k.

If ρ_k is close to 1, it is implied that the numerator and denominator of Equation (5) are roughly equal, which further implies that m_k is a good approximation of f at the current iterate. This allows the algorithm to increase the value of Δ_k for a subsequent iteration [3]. Finally, in the case where $\rho_k > 0$ but much less than 1, the value of Δ_k is unchanged. However, for $0 < \rho_k \ll 1$, the trust-region is shrunk [3]. Nocedal and Wright (2006) include a more thorough description of the general trust-region algorithm (Algorithm 1), however it is omitted in the interest of brevity.

B. An Overview of the Dogleg Method

The trust-region approach is commonly seen in many aspects of optimization today. In particular, the dogleg method is a trust-region method that can be employed when the symmetric matrix B_k is positive definite. This section seeks to outline the dogleg method, while referencing [3]. Note that for the remainder of this section, we focus on one iteration of a trust-region algorithm, as done in [3].

We begin by considering a single iteration of the subproblem given by Equation (4),

$$\min_{d \in X_k} m(d) \equiv f + g^T d + \frac{1}{2} p^T B_k d \qquad \text{s.t.} \quad \|d\| \leq \Delta\,, \tag{6}$$

from which the subscript k is dropped.

Letting $d^*(\Delta)$ be the solution to Equation (6), we consider then the case where B is positive definite [3]. This leads to the conclusion that $d^B = -B^{-1}g$ is the unconstrained minimizer of m [3]. Now if $d^B \in X_k$, it is a feasible point, from which it follows that $d^*(\Delta) = d^B$, for $\Delta \geq \|d^B\|$. There are then several cases to consider with respect to the size of Δ [3].

We first examine when Δ is small compared to d^B [3]. In this instance, the quadratic term for m is negligible in the solution for Equation (4), due to the restriction $\|d\| \leq \Delta$ [3]. For median values of Δ, $d^*(\Delta)$ follows a curved solution path, seen in Figure 2.

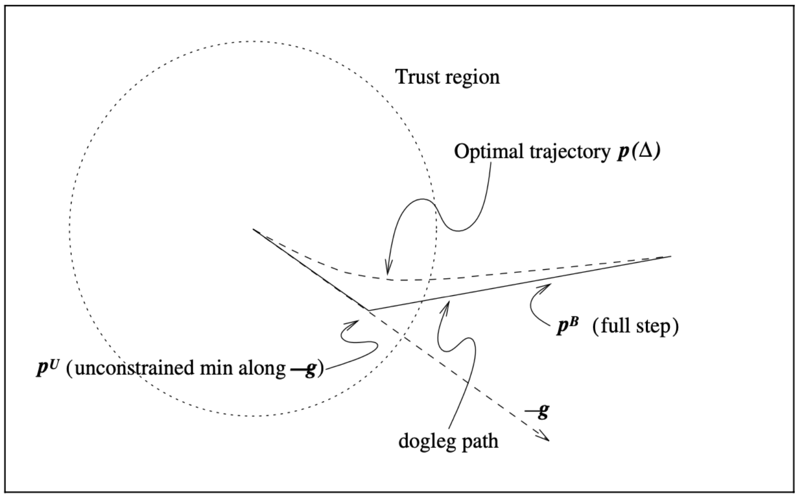
Trust region
Optimal trajectory $\boldsymbol{p}(\Delta)$
$\boldsymbol{p}^B$ (full step)
$\boldsymbol{p}^U$ (unconstrained min along $-\boldsymbol{g}$)
$-\boldsymbol{g}$
dogleg path

FIG. 2: Figure 4.4 from [3]. Here, a single iteration of the dogleg method is shown (note that $p = d$ in the figure). The steepest descent direction is given by $p^U = -(g^T g)g/(g^T Bg)$, and the second line segment runs from p^U to p^B. The trajectory for this line is given by $\tilde{p}(\tau)$, described in Equation (7). The solution point p is chosen to minimize the model m along this path, subject to the trust-region bound $\hat{\Delta}$.

As mentioned above, the trajectory of the dogleg path (line segment connecting p^U to p^B) is given by

$$\tilde{p}(\tau) = \begin{cases} \tau p^U\,, & 0 \le \tau \le 1, \\ p^U + (\tau - 1)(p^B - p^U)\,, & 1 \le \tau \le 2. \end{cases} \tag{7}$$

where $\tau \in [0, 2]$ [3]. Note that $p^U = d^U$ and $p^B = d^B$ here.

Now, when $\nabla^2 f(x_k)$ is positive definite, we can set

$$B = \nabla^2 f(x_k) \implies d^B = -(\nabla^2 f(x_k))^{-1} g_k\,,$$

in order to find the dogleg step [3]. If B is not positive definite, B can be chosen to be a modified Hessian obtained by adding a positive diagonal matrix or a full matrix to the true Hessian $\nabla^2 f(x_k)$ [3].

However, Nocedal and Wright (2006) argue that "use of a modified Hessian in the Newton-dogleg method is not completely satisfying from an intuitive viewpoint" [3]. This leads them to conclude that the dogleg method is best used when the Hessian $\nabla^2 f(x_k)$ is always positive semidefinite [3]. That is to say, the eigenvalues of the Hessian are all nonnegative.

C. The Retrospective Trust-Region Method

In this section, we detail the procedure of the Retrospective Trust-Region Method (RTRM), introduced by Bastin et al. (2007). The following mathematical theory is referenced from [1].

Referring to Section II A, recall that the trust-region method procedure involves determining a radius Δ_k during each iteration, and then modifying its size based on whether a model function m_k is successfully minimized [1]. The ratio ρ_k, given by Equation (5), is used to determine if the radius is increased or decreased. This process is repeated until the algorithm successfully optimizes the given objective function.

The above methodology describes a classical framework for trust-region algorithms [1]. The RTRM differs slightly from the above, in that it updates the radius after the k^{th} iteration, based on the value of the "retrospective" ratio,

$$\tilde{\rho}_{k+1} \equiv \frac{f(x_{k+1}) - f(x_{k+1} - s_k)}{m_{k+1}(x_{k+1}) - m_{k+1}(x_{k+1} - s_k)} = \frac{f(x_k) - f(x_k - s_k)}{m_{k+1}(x_k) - m_{k+1}(x_k - s_k)}\,, \tag{8}$$

where s_k is the step found by minimizing m_k within the trust-region, and the ratio is computed at the start of the $(k+1)^{th}$ iteration [1].

The RTRM method specifies Equation (5) as having two distinct roles, one where it accepts or rejects the new iterate and one where it computes the trust-region's radius [1]. This is unlike the classical trust-region framework.

The above mathematical theory, in conjunction with the established trust-region method, leads Bastin et al. (2007) to argue that their RTRM is "competitive" compared to optimization methods commonly used today [1].

III. A NUMERICAL COMPARISON OF TRUST-REGION METHODS

In this section, we outline a test suite used to compare various metrics between the dogleg method and the RTRM. In particular, we assess the accuracy of the final optimal point returned by each algorithm, the number of iterations taken to achieve the optimal point, and also the convergence rates of each algorithm for the various functions. Our test suite consists of the Rosenbrock, Sphere, and McCormick functions, all of which are benchmarks commonly used to evaluate the performance of optimization algorithms [4]. For $\mathbb{R}^2$, these functions take the form

$$f_1(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2\,, \tag{9}$$

$$f_2(x_1, x_2) = x_1^2 + x_2^2\,, \tag{10}$$

$$f_3(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 + 1\,, \tag{11}$$

where f_1, f_2, f_3 are the Rosenbrock, Sphere, and McCormick functions respectively, and their global minimizers are [4]

$$\mathbf{x}_1^* = (1, 1)^T \implies f_1(\mathbf{x}_1^*) = 0\,, \tag{12}$$

$$\mathbf{x}_2^* = (0, 0)^T \implies f_2(\mathbf{x}_2^*) = 0\,, \tag{13}$$

$$\mathbf{x}_3^* = (-0.54719, -1.54719)^T \implies f_3(\mathbf{x}_3^*) = -1.9133\,. \tag{14}$$

Note that the Rosenbrock and McCormick functions are both non-convex, while the Sphere function is convex [4]. These functions are deliberately chosen to introduce variety into the test suite.

We begin first with a treatment of the dogleg method for each of Equations (9), (10), and (11). Following the outline provided in Nocedal and Wright (2006), we implement the dogleg method in Python using the parameters $\Delta_0 = 0.5, \eta = 0.15$, and $\hat{\Delta} = 1$ (code for this is provided in the Appendix) [3]. Using an initial point of $x_0 = (3, -2.9)^T$ and a tolerance of 10^{-6}, we notice that the dogleg method converges close to the optimal point, within a small number of iterations. Table I shows the results of this experiment.

We then begin a treatment of the RTRM using the same test suite as above. Following the outlined Algorithm 2.1 from Bastin et. al (2007), we implement the RTRM in Python using the parameters $\eta_1 = 0.15, \tilde{\eta}_1 = 0.9, \tilde{\eta}_2 = 0.9, \gamma_1 = 0.9$, and $\gamma_2 = 0.9$ (code for this is provided in the Appendix) [1]. These constants help the algorithm determine the value of the trust region radius. Using again a tolerance of 10^{-6} and an initial point of $x_0 = (3, -2.9)^T$, we find that the RTRM also converges close to the optimal points for each function, albeit with a consistently larger amount of iterations. Table II details the results of this experiment.

Function Name	Optimal Point $x_{opt,dogleg}$	# of Iterations
Rosenbrock	$(1, 1)^T$	23
Sphere	$(-2.5451 \times 10^{-10}, 2.4603 \times 10^{-10})^T$	5
McCormick	$(-0.54719, -1.54719)^T$	7

TABLE I: Results from optimizing our test suite of benchmark functions using the dogleg trust-region method. Note that the value of $x_{opt,dogleg}$ for the sphere function is essentially $(0, 0)^T$, and observe that the values for $x_{opt,dogleg}$ for each function exactly match those given in Equations (12), (13), and (14). The number of iterations required to reach the optimal points are all relatively low compared to the results from the RTRM.

Function Name	Optimal Point $x_{opt,rtrm}$	# of Iterations
Rosenbrock	$(0.99999, 0.99999)^T$	15395
Sphere	$(-6.8637 \times 10^{-8}, 6.6350 \times 10^{-8})^T$	79
McCormick	$(-0.54719, -1.54719)^T$	115

TABLE II: Results from optimizing our test suite of benchmark functions using the retrospective trust-region method. Note that the values of $x_{opt,rtrm}$ for the Rosenbrock and sphere functions are essentially $(1, 1)^T$ and $(0, 0)^T$ respectively. The values for $x_{opt,rtrm}$ for each function exactly match those given in Equations (12), (13), and (14). The number of iterations required to reach the optimal points are very high relative to the results from the dogleg method.

Following the collection of results concerning the accuracy of the optimal points and the number of iterations taken to obtain them, we then assess the convergence rates for each algorithm. This is done by recording the values of the objective functions (Equations (9), (10), and (11)) during each iteration and plotting them against the number of iterations the algorithms run for. The minimum values of each function are given in Equations (12), (13), (14). Results of the convergence rate comparisons are shown in Figures 3, 4, and 5. We provide an analysis of these convergence results, along with the results from Tables I and II, in Section IV.

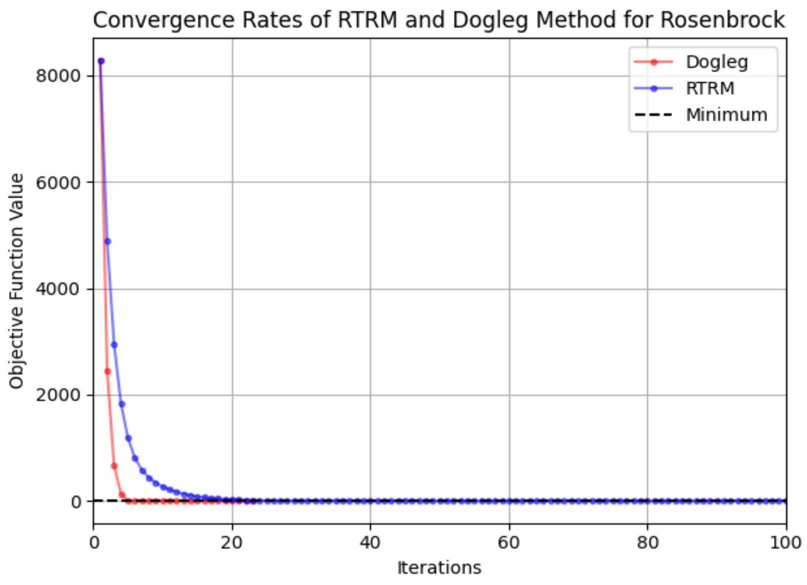

FIG. 3: A plot of convergence rates calculated using the value of $f_1(x_k)$ for the Rosenbrock function, where k is the number of iterations. The dogleg and RTRM algorithms are each used for the comparison. The expected minimum value of the Rosenbrock function $f_1(\mathbf{x}_1^*) = 0$ is shown by a black dotted line. Note that the x-axis is limited to the first 100 iterations, so that the dogleg's datapoints can be seen.

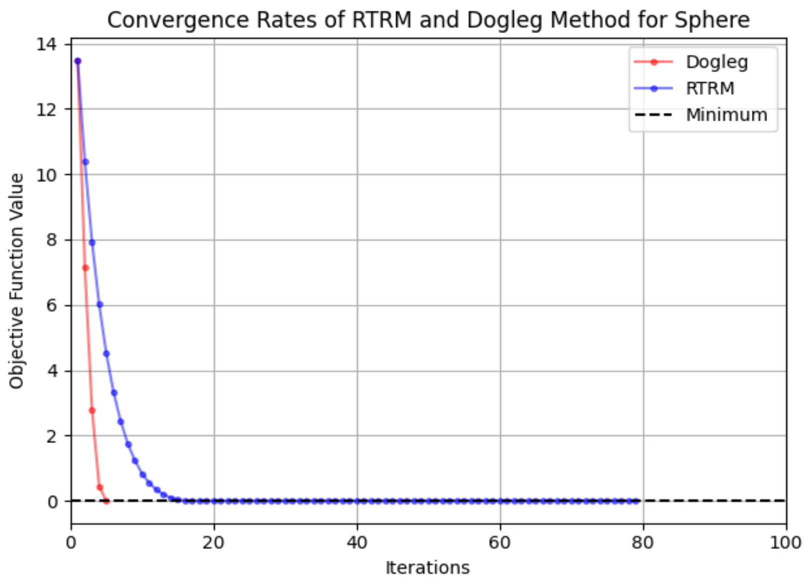

FIG. 4: A plot of convergence rates calculated using the value of $f_2(x_k)$ for the Sphere function, where k is the number of iterations. The dogleg and RTRM algorithms are each used for the comparison. The expected minimum value of the Sphere function $f_2(\mathbf{x}_2^*) = 0$ is shown by a black dotted line. Note that the x-axis is limited to the first 100 iterations, so that the dogleg's datapoints can be seen.

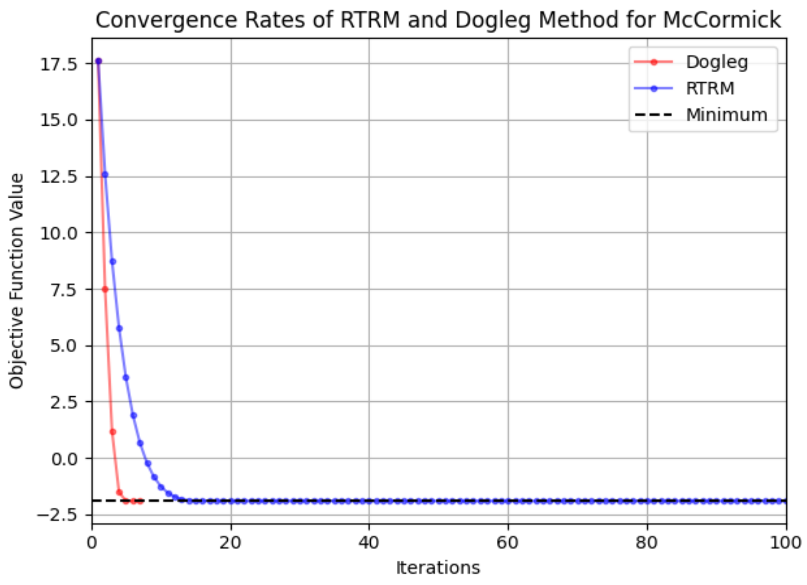

FIG. 5: A plot of convergence rates calculated using the value of $f_3(x_k)$ for the McCormick function, where k is the number of iterations. The dogleg and RTRM algorithms are each used for the comparison. The expected minimum value of the McCormick function $f_3(\mathbf{x}_3^*) = -1.9133$ is shown by a black dotted line. Note that the x-axis is limited to the first 100 iterations, so that the dogleg's datapoints can be seen.

IV. DISCUSSION

In this section, we discuss the significance and meaning of the results from Section III.

An immediate trend we notice from Tables I and II is while both the dogleg method and RTRM converge to almost exactly the optimal points described in Equations (12), (13), and (14), there is a significant difference in the number of iterations taken to reach their goal. In particular, the dogleg method greatly outperforms the RTRM in the number of iterations it takes to reach the optimal point for each test function. This discrepancy is also shown in Figures 3, 4, and 5, where we directly measure the performance of each algorithm's convergence.

There are many possible reasons for this disparity in the number of iterations. To start, we note that the test functions chosen might not vary significantly in complexity, which could make it easier for the well-established dogleg method to optimize them in fewer iterations. Specifically, the Sphere and Rosenbrock functions are heavily-tested benchmarks that are well-defined around their minimum, potentially making it easier for the dogleg method to optimize.

In addition, the RTRM depends on several constants that must be chosen prior to testing, as opposed to the dogleg method. Each of these constants strongly influence the efficacy of the RTRM algorithm, so the freedom of choice in this case may have negatively affected the RTRM's performance.

We can then consider the nature of each algorithm's trust-region radius update method. Since the RTRM updates its trust-region radius using the previous iteration instead of the current, it is prone to making "safer" choices that might not allow rapid convergence. In contrast, the dogleg method only considers the current iteration for its trust-region radius update, leading it to potentially make bolder choices for subsequent radii.

Finally, an important possibility to consider is human error. As the RTRM is a new and complex optimization algorithm that is more intensive than the dogleg method, there is a potential for oversight in its implementation.

Nevertheless, the results of our experiments do not necessarily imply that the RTRM is an inferior method compared to the dogleg. Since the RTRM is nearly as accurate as the dogleg method in terms of computing the optimal points, it still offers a competitive way of optimizing numerical problems. Further testing, with a variety of problems, could address the optimizing potential of the RTRM.

V. CONCLUSIONS

In this paper, we analyze and compare the performance of the dogleg trust-region method to that of a recently developed algorithm in optimizing several objective functions. By referencing various works of literature, we choose to test the efficacy of the retrospective trust-region method, developed by Bastin et al. (2007), against the dogleg method [1].

To begin this comparison, we first reference Nocedal and Wright (2006) and Bastin et al. (2007) in order to strengthen our understanding of the underlying theory involved in both algorithms. We discuss this theory in Section II A by first covering the fundamentals of the trust-region method, and then by outlining the procedures used to implement each of the dogleg and retrospective trust-region methods [1, 3]. In Section III, we implement these algorithms in Python and determine their accuracy in computing the optimal points of Equations (9), (10), (11), and also record the number of iterations each method takes during the optimization process. Figures 3, 4, and 5 were created to display the convergence rates of each algorithm when used to optimize the test suite functions.

We then conduct an analysis of results in Section IV. It was found that both the dogleg method and the RTRM converge close to the optimal points, but the RTRM method takes significantly more iterations. Despite this finding, we agree with the conclusion made by Bastin et al. (2007), in that the RTRM is a competitive optimization algorithm for both convex and non-convex problems.

Future studies should involve more numerical experiments that test other function types as benchmarks, and that use alternate initial values for the constants. Specifically, more research might yield methods for choosing the values of the RTRM more rigorously, potentially allowing for more rapid convergence results.

Appendix

In this section, we provide the code used to implement the dogleg method and the RTRM, as well as the plotting code used to create the various figures. We extensively follow the procedure outlined by Nocedal and Wright (2006) and Bastin et al. (2007) in order to implement the `dogleg` and `rtrm` functions [1, 3]. Note that the constants $\eta_1 = 0.15, \tilde{\eta}_1 = 0.9, \tilde{\eta}_2 = 0.9, \gamma_1 = 0.9$, and $\gamma_2 = 0.9$ for `rtrm` are chosen arbitrarily according to the bounds set in Bastin et al. (2007) [1]. Future studies might be able to find a way to make these choices more methodological. Also, the values of the Δ_k in Step 2 of `rtrm` are chosen to be the minimum of the bounds outlined by Bastin et al. (2007) [1].

```python
import numpy as np
import matplotlib.pyplot as plt

def rosenbrock(x):
    return 100 * (x[1] - x[0]**2)**2 + (1 - x[0])**2

def rosenbrock_gradient(x):
    return np.array([-400 * x[0] * (x[1] - x[0]**2) - 2 * (1 - x[0]), 200 * (x[1]
    - x[0]**2)])

def sphere(x):
    return x[0]**2 + x[1]**2

def sphere_gradient(x):
    return np.array([2 * x[0], 2 * x[1]])

def mccormick(x):
    return np.sin(x[0] + x[1]) + (x[0] - x[1])**2 - 1.5 * x[0] + 2.5 * x[1] + 1

def mccormick_gradient(x):
    return np.array([np.cos(x[0] + x[1]) + 2 * (x[0] - x[1]) - 1.5,
                     np.cos(x[0] + x[1]) - 2 * (x[0] - x[1]) + 2.5])

def hessian_approx(x, f):  # from Homework 2
    h = 10**(-8)
    b_k = np.empty([2, 2])

    e_1 = np.array([1, 0])
    e_2 = np.array([0, 1])

    b_k[0, 0] = (f(x + h * e_1)[0] - f(x)[0]) / h
    b_k[1, 0] = (f(x + h * e_2)[0] - f(x)[0]) / h
    b_k[0, 1] = (f(x + h * e_1)[1] - f(x)[1]) / h
    b_k[1, 1] = (f(x + h * e_2)[1] - f(x)[1]) / h

    return b_k

def m_k(f, grad_f, B, p):
    return f + np.dot(grad_f, p) + 0.5 * np.dot(p, np.dot(B, p))

def dogleg_method(f, grad_f, x0, delta_hat, tol=1e-6, delta=0.5, eta=0.15):
    x = np.array(x0)
    k = 0
    f_values = []  # value of f at each iteration

    while np.linalg.norm(grad_f(x)) > tol:
        g = grad_f(x)
        B = hessian_approx(x, grad_f)

        p_B = -np.dot(np.linalg.inv(B), g)  # Newton step
        if np.linalg.norm(p_B) <= delta:
            p = p_B
        else:
            p_U = -np.dot(g, g) / np.dot(g, np.dot(B, g)) * g  # Steepest descent
step
            if np.linalg.norm(p_U) >= delta:
                p = delta / np.linalg.norm(p_U) * p_U
            else:
                # Dogleg step
                a = np.dot(p_B - p_U, p_B - p_U)
                b = 2 * np.dot(p_B - p_U, p_U)
                c = np.dot(p_U, p_U) - delta**2
                # solving scalar quadratic on page 75 of Nocedal
                tau = (-b + np.sqrt(b**2 - 4 * a * c)) / (2 * a)
                p = p_U + tau * (p_B - p_U)

        rho = (f(x) - f(x + p)) / (m_k(f(x), grad_f(x), B, [0, 0]) - m_k(f(x),
grad_f(x), B, p))

        if rho < 0.25:
            delta = 0.25 * delta
        elif rho > 0.75 and np.linalg.norm(p) == delta:
            delta = min(2 * delta, delta_hat)

        if rho > eta:
            x = x + p

        k += 1
        f_values.append(f(x))

    return x, k, f_values

def rtrm(f, grad_f, x_0, delta_0=0.5, eta_1=0.15, eta_tilde_1=0.9,
         eta_tilde_2=0.9, gamma_1=0.9, gamma_2=0.9, tol=1e-6):
    x = x_0
    delta_k = delta_0
    k = 0
    f_values = []  # value of f at each iteration
```

```python
    x_prev = None
    delta_prev = None

    while np.linalg.norm(grad_f(x)) > tol:
        # Step 1: Model definition
        fx = f(x)
        grad_fx_k = grad_f(x)
        B_k = hessian_approx(x, grad_f)

        # Step 2: Retrospective trust-region radius update
        if k > 0:
            if np.all(x == x_prev):
                # delta_k = np.random.uniform(gamma_1 * delta_prev, gamma_2 *
    delta_prev)
                delta_k = gamma_1 * delta_prev
            else:
                rho_tilde_k = (f(x_prev) - fx) / \
                              (m_k(f(x_prev), grad_f(x_prev), B_k, x_prev - x)
                               - m_k(fx, grad_fx_k, B_k, x_prev - x))
                if rho_tilde_k >= eta_tilde_2:
                    delta_k = delta_prev
                elif eta_tilde_1 <= rho_tilde_k < eta_tilde_2:
                    delta_k = gamma_2 * delta_prev
                else:
                    delta_k = gamma_1 * delta_prev

        # Step 3: Step calculation
        s_k = -delta_k * grad_fx_k / np.linalg.norm(grad_fx_k)

        # Step 4: Acceptance of the trial point
        rho_k = (fx - f(x + s_k)) / \
                (m_k(fx, grad_fx_k, B_k, np.zeros_like(x))
                 - m_k(fx, grad_fx_k, B_k, s_k))
        if rho_k > eta_1:
            x_next = x + s_k
        else:
            x_next = x

        x_prev = x
        x = x_next
        delta_prev = delta_k
        k += 1
        f_values.append(f(x))

    return x, k, f_values

x0 = np.array([3.0, -2.9])  # Initial point
delta_bound = 1
result_dog_b, num_dog_b, values_dog_b = dogleg_method(rosenbrock,
                                                     rosenbrock_gradient, x0, delta_bound)
result_dog_s, num_dog_s, values_dog_s = dogleg_method(sphere,
                                                     sphere_gradient, x0, delta_bound)
result_dog_m, num_dog_m, values_dog_m = dogleg_method(mccormick,
                                                     mccormick_gradient, x0, delta_bound)

# results for dogleg
print("Optimal point for Rosenbrock using Dogleg:", result_dog_b)
print("Number of iterations for Rosenbrock using Dogleg:", num_dog_b)
print("Optimal point for Sphere using Dogleg:", result_dog_s)
print("Number of iterations for Sphere using Dogleg:", num_dog_s)
print("Optimal point for McCormick using Dogleg:", result_dog_m)
print("Number of iterations for McCormick using Dogleg:", num_dog_m)


# results for rtrm
result_rtrm_b, num_rtrm_b, values_rtrm_b = rtrm(rosenbrock, rosenbrock_gradient, x0)
result_rtrm_s, num_rtrm_s, values_rtrm_s = rtrm(sphere, sphere_gradient, x0)
result_rtrm_m, num_rtrm_m, values_rtrm_m = rtrm(mccormick, mccormick_gradient, x0)
print("Optimal point for Rosenbrock using RTRM:", result_rtrm_b)
print("Number of iterations for Rosenbrock using RTRM:", num_rtrm_b)
print("Optimal point for Sphere using RTRM:", result_rtrm_s)
print("Number of iterations for Sphere using RTRM:", num_rtrm_s)
print("Optimal point for McCormick using RTRM:", result_rtrm_m)
print("Number of iterations for McCormick using RTRM:", num_rtrm_m)

# comparing convergence rates for each function

# Rosenbrock
plt.figure(1)
plt.plot(np.arange(1, num_dog_b+1), values_dog_b, label='Dogleg', marker='.', color='red', alpha=0.5)
plt.plot(np.arange(1, num_rtrm_b+1), values_rtrm_b, label='RTRM', marker='.', color='blue', alpha=0.5)
plt.axhline(y=0, color='black', linestyle='--', label='Minimum')
plt.xlabel('Iterations')
plt.ylabel('Objective Function Value')
plt.xlim(0, 100)
plt.title('Convergence Rates of RTRM and Dogleg Method for Rosenbrock')
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig('rosenbrock comparison.png')
plt.show()

# Sphere
```

```
plt.figure(2)
plt.plot(np.arange(1, num_dog_s+1), values_dog_s, label='Dogleg', marker='.',
    color='red', alpha=0.5)
plt.plot(np.arange(1, num_rtrm_s+1), values_rtrm_s, label='RTRM', marker='.',
    color='blue', alpha=0.5)
plt.axhline(y=0, color='black', linestyle='--', label='Minimum')
plt.xlabel('Iterations')
plt.ylabel('Objective Function Value')
plt.xlim(0, 100)
plt.title('Convergence Rates of RTRM and Dogleg Method for Sphere')
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig('sphere comparison.png')
plt.show()

# McCormick
plt.figure(3)
plt.plot(np.arange(1, num_dog_m+1), values_dog_m, label='Dogleg', marker='.',
    color='red', alpha=0.5)
plt.plot(np.arange(1, num_rtrm_m+1), values_rtrm_m, label='RTRM', marker='.',
    color='blue', alpha=0.5)
plt.axhline(y=-1.9133, color='black', linestyle='--', label='Minimum')
plt.xlabel('Iterations')
plt.ylabel('Objective Function Value')
plt.xlim(0, 100)
plt.title('Convergence Rates of RTRM and Dogleg Method for McCormick')
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig('mccormick comparison.png')
plt.show()
```

REFERENCES

[1] F. Bastin, V. Malmedy, M. Mouffe, P. Toint, and D. Tomanos. A retrospective trust-region method for unconstrained optimization. *Mathematical Programming*, 123:395–418, 06 2007.

[2] R. Hauser. Lecture 7 notes on continuous optimization, 2005.

[3] J. Nocedal and S. Wright. *Numerical optimization.* Springer, 2006.

[4] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved May 4, 2023, from `http://www.sfu.ca/~ssurjano`.

[5] Y.-x. Yuan. Recent advances in trust region algorithms. *Mathematical Programming*, 151, 06 2015.