كلية الدراسات العليا

Sudan University of Science and Technology
College of Graduate Studies

جامعة السودان للعلوم والتكنولوجيا

New Direction in Derivative Free Optimization

اتجاه جديد في الامثلية خالية المشتقة

A Thesis Submitted in Fulfillment of the Requirements for the Degree of Ph.D in Mathematics

By
Ahmed Mohamed Nageeb Alwasela ahmed

Supervisor
Dr.Mohsin Hassan Abdalla Hashim

2023

Dedications

To My Great Father, Mother, Brothers and Sister.

Acknowledgements

I would like to express deepest gratitude to my supervisor Dr. Mohsin Hassan Abdalla Hashim of University of Khartoum for his full support, expert guidance, understanding and encouragement throughout my study and research. A special thanks is extended to Sudan University of Science &Technology for giving me this chance for higher studies. Lastly my heart is full of thanks to my brothers and sister who helped in preparing this project and all dear friends for their help.

A special thanks is extended to D:Salwa Harfi Wadie he advised me to direct me to the right path and supported me academically and spare me any information whenever I needed help at any time in the search were supportive of me .

Abstract

In this thesis, we study a derivative-free trust-region algorithm for large-scale unconstrained optimization, using symmetric-rank1 (SR1) to update the Hessian at every iteration. The centeral finite-difference iterations are used to approximate the gradient of the function. The iterative solution method and truncated Newton method were used to solve the trust-region sub-problem. Its performance is tested on some problems and compared the solutions found by truncated Newton method and iterative solution method.

الخلاصة

في هذه الاطروحة قمنا بدراسة خوارزمية منطقة - الثقة خالية المشتقة للأمثلية غير المقيدة علي نطاق واسع وباستخدام الرتبة (SR1) لتحديث هيسيان في كل تكرار. تم استخدام تكرارات الفرق المنتهي المركزي لتقريب انحدار الدالة .طريقة الحل التكراري وطريقة نيوتن المقطوعة تم استخدامهما لحل المسألة الجزئية لمنطقة- الثقة. تم اختبار ادائها علي بعض المسائل ومقارنة الحلول التي وجدت بواسطة طريقة نيوتن المقطوعة وطريقة الحل التكراري .

The Contents

Chapter 1

1.1 Introduction:

Many industrial and engineering applications need to solve optimization problems in which the derivatives of the objective function are unavailable. They try to avoid unnecessary evaluations in the objective function. The absences of computable derivatives prohibit the use of Taylor models largely used in differentiable problems. Moreover, in general, the optimization without derivatives is not easy, since we attempt to obtain a minimum point with less information [5].Derivative free optimization (DFO) methods are designed for solving nonlinear optimization without constraints where the derivative of the objective function are not available. We consider formally the problem

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

Where $f$is a smooth nonlinear objective function from R^n into R and is bounded below. We assume that the gradient $\nabla f(x)$ and the Hessian $\nabla^2 f(x)$ can not be computed for any x. There is a high demand from practitioners for such methods because this kind of problems occur relatively frequently in the industry. In applications either the evaluation of the objective function $f(x)$ is very difficult or expensive, or the derivatives of $\boldsymbol{f}$are not available. The last situation occurs when the computation of $\boldsymbol{f(x)}$ at a given point $\boldsymbol{x}$ results from some physical, chemical or econometrical experiment or measurement, or is a result of large and expensive computer simulation for which the source code is either not available or un modifiable, which can be considered as a black box. In practice the value of $f(x)$ is contaminated with noise or may be non-smooth; but we don't consider these cases in this proposal.

There are mainly four classes of derivative-free optimization methods.

The first class of DFO algorithms are the direct search or pattern search methods which are based on the exploration of the variable space by using sample points from a predefined class geometric pattern and use either the Melder-Need simplex algorithm or parallel direct search algorithm.They do not exploit the inherit smoothness of the objective function and require therefore a very large number of function evaluations. They can be useful for non-smooth problems.

The second class of DFO's are line search methods which consists of a sequence of $n + 1$ one-dimensional searches introduced by Powell. The combination of finite difference techniques coupled with quasi-Newton method constitutes the third class of the algorithms. The last class of the methods are based on modeling the objective function by multivariate interpolation in combination with the trust-region techniques.These methods were introduced by Winfried and by Powell. In this search we consider this class of DFO algorithms [1,10,12].

1.2 Statement of the problem:

Designing a method to solve

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

Without the use of derivatives. it improves the way the derivatives are approximated .

1.3 Objectives:

- Attaining efficiency in the solution of the problem.
- Enhancing trust region based methods.
- Studying the performance of the proposed method.

1.4 Methodology:

- Polynomial interpolation is used to approximate the function $f(x)$ at a points $\{y^1, y^2, y^3, \dots \dots, y^N\}$ where $N = \frac{(n+1)(n+2)}{2}$ .These points are randomly selected using a proposed technique..
- trust region method is used in every iteration to produce an acceptable decrease in the function value.
- Merging the search-based method for enhancement.

1.5 Thesis Layout:

In Chapter 2 Fundamentals of Unconstrained Optimization. Line Search and Trust Regions. Some well-known method are studied.

Chapter 3 Trust Region and solution of its sub-problem. Details of Quasi-Newton's Methods in Chapter 4.

Chapter 5 discusses the main algorithms and results. MATLAB Computational of the Proposed Algorithm and The thesis Conclusion.

Chapter 2
Fundamentals of Unconstrained Optimization

2.1 Introduction:

In unconstrained optimization, we minimize an objective function that depends on real variables, with no restrictions at all on the values of these variables. The mathematical formulation is

$$min_x f(x) \qquad \textbf{(2.1)}$$

Where$x \in IR^n$is a real vector with $n \geq 1$ components and $f: IR^n \rightarrow IR$is a smooth function.Usually, we lack a global perspective on the function$f$. All we know are the values of f and maybe some of its derivatives at a set of points $x_0, x_1, x_2, \ldots$.fortunately, our algorithms get to choose these points, and they try to do so in a way that identifies a solution reliably and without using too much computer time or storage. Often, the information about $f$does not come cheaply, so we usually prefer algorithms that do not call for this information unnecessarily[1,12].

2.2 What is a Solution?

Generally, we would be happiest if we found a global minimizer of f , a point where the function attains its least value. A formal definition is a point x^* is a global minimizer if $f(x^*) \leq f\ (x)$ for all x, where x ranges over all of IR^n (or at least over the domain of interest to the modeler). The global minimizer can be difficult to find, because our knowledge of f is usually only local.Since our algorithm does not visit many points, we usually do not have a good picture of the overall shape of f , and we can never be sure that the function does not take a sharp dip in some region that has not been sampled by the algorithm. Most algorithms are able to find only a local minimizer, which is a point that achieves the smallest value of f in its neighborhood. Formally, we say:

A point x^* is a local minimizer if there is a neighborhood γ of x^* such that $f\ (x^*) \leq f\ (x)$ for all $x \in \gamma$.(Recall that a neighborhood of x^* is simply an open set that contains x^*) A point that satisfies this definition is sometimes called a weak local minimizer. This terminology distinguishes it from a strict local minimizer, which is the outright winner in its neighborhood. Formally, A point x^* is a strict local minimizer (also called a strong local minimizer) if there is a neighborhood γ of x^* such that $f\ (x^*) < f(x)$ for all $x \in \gamma$ with $x \neq x^*$.For the constant function $f(x) = 2$, every point x is a weak local minimizer, while the function $f(x) = (x - 2)^4$ has a strict local minimizer at $x = 2$.A slightly more exotic type of local minimizer is defined as follows.

A point $x^*$is an isolated local minimizer if there is a neighborhood γ of x^* such that $x^*$is the only local minimizer in γ.Some strict local minimizers are not isolated, as illustrated by the function

$$f(x) = x^4 \cos\left(\frac{1}{x}\right) + 2x^4\,, f(0) = 0$$

Which is twice continuously differentiable and has a strict local minimizer at $x^* = 0$.

However, there are strict local minimizers at many nearby points x_j , and we can label these points so that $x_j \to 0$ as $j \to \infty$.While strict local minimizers are not always isolated, it is true that all isolated local minimizers are strict.

Figure 2.1 illustrates a function with many local minimizers. It is usually difficult to find the global minimizer for such functions, because algorithms tend to be "trapped"at local minimizers. This example is by no means pathological. In optimization problems associated with the determination of molecular conformation, the potential function to be minimized may have millions of local minima.

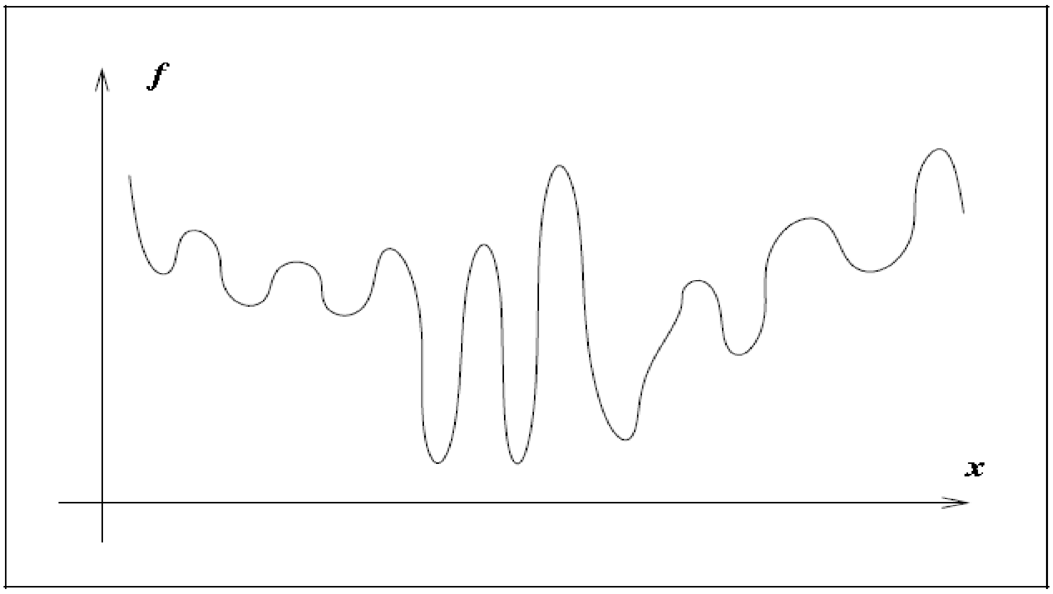


Figure 2.1 A difficult case for Global Minimization.

Sometimes we have additional "global" knowledge about f that may help in identifying global minima. An important special case is that of convex functions, for which every local minimizer is also a global minimizer.

Recognizing a Local Minimum:

From the definitions given above, it might seem that the only way to find out whether a point x^* is a local minimum is to examine all the points in its immediate vicinity, to make sure that none of them has a smaller function value. When the function $\boldsymbol{f}$ is smooth, however, there are more efficient and practical ways to identify local minima. In particular, if $\boldsymbol{f}$ is twice continuously differentiable, we may be able to tell that x^* is a local minimizer (and possibly a strict local minimizer) by examining just the gradient $\nabla f(x^*)$ and the Hessian $\nabla^2 f(x^*)$.The mathematical tool used to study minimizers of smooth functions is Taylor's theorem. Because this theorem is central to our analysis throughout the search, we state it now.

Theorem 2.1 (Taylor's Theorem).

Suppose that $f: IR^n \rightarrow IR$ is continuously differentiable and that $p \in IR^n$. Then we have that

$$f(x+p) = f(x) + \nabla f(x+tp)^T p \qquad \textbf{(2.2)}$$

For some $t \in (0,1)$. Moreover, if f is twice continuously differentiable, we have that

$$\nabla f(x+p) = \nabla f(x) + \int_0^1 \nabla^2 f(x+tp)p dt \qquad \textbf{(2.3)}$$

and that

$$f(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x+tp)p, \qquad \textbf{(2.4)}$$

For some $t \in (0,1)$.

Necessary conditions for optimality are derived by assuming that x^* is a local minimize and then proving facts about $\nabla f(x^*)$ and $\nabla^2 f(x^*)$.

Theorem 2.2 (First-Order Necessary Conditions):

If x^* is a local minimizer and f is continuously differentiable in an open neighborhood of x^*, then $\nabla f(x^*) = 0$.

Proof.

Suppose for contradiction that $\nabla f(x^*) \neq 0$. Define the vector $p = -\nabla f(x^*)$ and note that $p^T \nabla f(x^*) = -\|\nabla f(x^*)\|^2 < 0$. Because ∇f is continuous near x^*, there is ascalar $T > 0$ such that

$p^T \nabla f(x^* + tp) < 0$, for all $t \in [0, T]$.For any $\bar{t} \in (0, T]$, we have by Taylor's theorem that

$f(x^* + \bar{t}p) = f(x^*) + \bar{t}p^T \nabla f(x^* + tp)$, for some $t \in (0, \bar{t})$.

Therefore, $f(x^* + \bar{t}p) < f(x^*)$for all $\bar{t} \in (0, T]$. We have found a direction leading away from x^* along which f decreases, so x^* is not a local minimizer, and we have acontradiction. We call x^* a stationary point if $\nabla f(x^*) = 0$. According to Theorem 2.2, any local minimizer must be a stationary point.

For the next result we recall that a matrix $\boldsymbol{B}$ is positive definite if $p^T B p > 0$ for all $p \neq 0$, and positive semi definite if $p^T B p \geq 0$ for all p.

Theorem 2.3 (Second-Order Necessary Conditions):

If $x^*$is a local minimizer of $\boldsymbol{f}$ and $\nabla^2 f$ exists and is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semi definite.

Proof.

We know from Theorem 2.2, that $\nabla f(x^*) = 0$. For contradiction, assume that $\nabla^2 f(x^*)$ is not positive semi definite. Then we can choose a vector p such that $p^T \nabla^2 f(x^*)p < 0$, and because $\nabla^2 f$ is continuous near x^*, there is a scalar $T > 0$ such that $p^T \nabla^2 f(x^* + tp)p < 0$ for all $t \in [0, T]$.

By doing a Taylor series expansion around x^*, we have for all $\bar{t} \in (0, T]$ and some $t \in (0, \bar{t})$ that

$f(x^* + \bar{t}p) = f(x^*) + \bar{t}\, p^T \nabla f(x^*) + \frac{1}{2}\bar{t}^2 p^T \nabla^2 f(x^* + tp)p < f(x^*)$.

As in Theorem 2.2, we have found a direction from x^* along which f is decreasing, and so again, x^* is not a local minimize.

Theorem 2.4 (Second-Order Sufficient Conditions):

Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.

Theorem 2.5:

When f is convex, any local minimize x^* is a global minimizer of f. If in addition f is differentiable, then any stationary point x^* is a global minimizer of f .

Proof.

Suppose that x. is a local but not a global minimizer. Then we can find a point $z \in IR^n$ with $f(z) < f(x^*)$. Consider the line segment that joins x^* to z, that is,

$$x = \lambda z + (1 - \lambda)x^* \text{ ,for some } \lambda \in (0, 1]. \qquad \textbf{(2.5)}$$

by the convexity property for f , we have

$$f(x) \leq \lambda f(z) + (1 - \lambda)f(x^*) < f(x^*) \qquad \textbf{(2.6)}$$

any neighborhood N of x. contains a piece of the line segment (2.5), so there will always be points $x \in N$ at which (2.6) is satisfied. Hence, x^* is not a local minimizer.For the second part of the theorem, suppose that x^* is not a global minimizer and choose z as above. Then, from convexity, we have

$$\nabla f(x^*)^T (z - x^*) = \frac{d}{d\lambda} f\big(x^* + \lambda(z - x^*)\big) \,|_{\lambda=0}$$

$$\lim_{\lambda \downarrow 0} \frac{f\big(x^* + \lambda(z - x^*)\big) - f(x^*)}{\lambda} \leq \lim_{\lambda \downarrow 0} \frac{\lambda f(z) + (1 - \lambda)f(x^*) - f(x^*)}{\lambda}$$

$f(z) - f(x^*) < 0$.

Therefore, $\nabla f(x^*) \neq 0$, and so x^* is not a stationary point.

Nonsmooth problems:

This search focuses on smooth functions, by which we generally mean functions whose second derivatives exist and are continuous. We note, however, that there are interesting problems in which the functions involved may be nonsmooth and even discontinuous. It is not possible in general to identify a minimizer of a general discontinuous function. If, however, the function consists of a few smooth pieces, with discontinuities between the pieces, it may be possible to find the minimizer by minimizing each smooth piece individually. If the function is continuous everywhere but non-differentiable at certain points, as in Figure 2.2, we can identify a solution by examing the subgradient or generalized gradient, which are generalizations of the concept of gradient to the nonsmooth case[1,12].

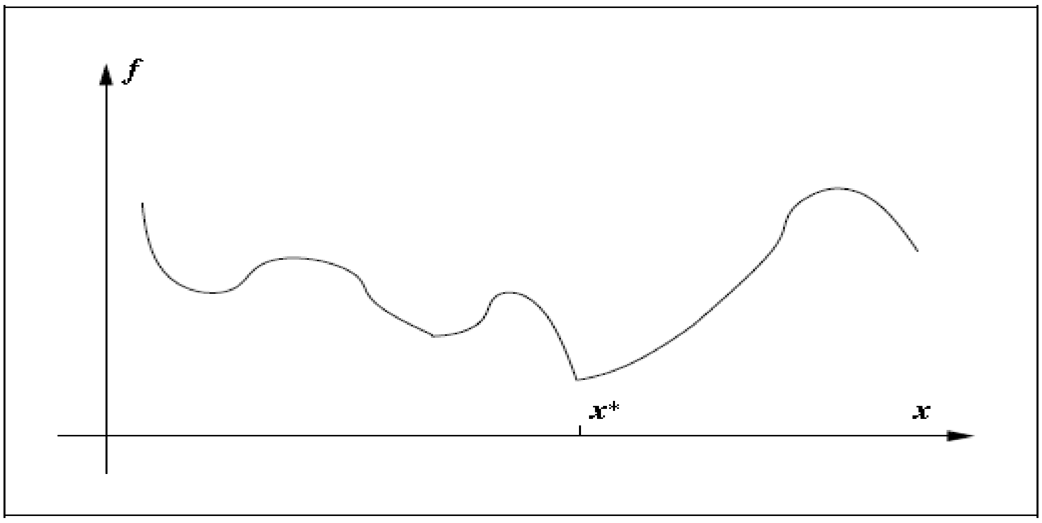


Figure 2.2 Nonsmooth function with minimum at a kink.

Here, we mention only that the minimization of a function such as the one illustrated in Figure 2.2(which contains a jump discontinuity in the first derivative $f^{'}(x)$ at the minimum) is difficult because the behavior of f is not predictable near the point of nonsmoothness. That is, we cannot be sure that information about f obtained at one point can be used to infer anything about f at neighboring points, because points of nondifferentiability may intervene. However, minimization of certain special nondifferentiable functions, such as

$$f(x) = \|r(x)\|_1 , f(x) = \|r(x)\|_\infty \qquad \textbf{(2.7)}$$

(where$r(x)$ is a vector function),

2.3 Overview of Algorithms:

All algorithms for unconstrained minimization require the user to supply a starting point, which we usually denote by x_0. The user with knowledge about the application and the data set may be in a good position to choose x_0 to be a reasonable estimate of the solution. Otherwise, the starting point must be chosen by the algorithm, either by a systematic approach or in some arbitrary manner.

Beginning at x_0, optimization algorithms generate a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ that terminate when either no more progress can be made or when it seems that a solutionpoint has been approximated with sufficient accuracy. In deciding how to move from one iterate x_k to the next, the algorithms use information about the function f at x_k , and possibly also information from earlier iterates $x_0, x_1, \ldots, x_{k-1}$. They use this information to find a new iterate x_{k+1} with a lower function value than x_k. (There exist nonmonotone algorithms that do not insist on a decrease in f at every step, but even these algorithms require f to be decreased after some prescribed number m of iterations, that is, $f(x_k) < f\ (x_{k-m})$.

There are two fundamental strategies for moving from the current point x_k to a new iterate x_{k+1}. Most of the algorithms described in this search follow one of these approaches.

Two Strategies: Line Search and Trust Region:

In the line search strategy, the algorithm chooses a direction p_k and searches along this direction from the current iterate x_k for a new iterate with a lower function value. The distance to move along p_k can be found by approximately solving the following one dimensional minimization problem to find a step length α:

$$min_{\alpha>0} f(x_k + \alpha p_k) \qquad \textbf{(2.8)}$$

By solving **(2.8)** exactly, we would derive the maximum benefit from the direction p_k, but an exact minimization may be expensive and is usually unnecessary. Instead, the line search algorithm generates a limited number of trial step lengths until it finds one that loosely approximates the minimum of **(2.8)**. At the new point, a new search direction and step length are computed, and the process is repeated.

In the second algorithmic strategy, known as trust region, the information gathered about f is used to construct a model function m_k whose behavior near the current point x_k is similar to that of the actual objective function . Because the model m_k may not be agood approximation of f when x is far from x_k , we restrict the search for a minimizer of m_k to some region around x_k . In other words, we find the candidate step p by approximately solving the following subproblem:

$$^{min}{}_{p} m_k(x_k + p), \text{ where } x_k + p \text{ lies inside the trust region.} \qquad \textbf{(2.9)}$$

If the candidate solution does not produce a sufficient decrease in f , we conclude that the trust region is too large, and we shrink it and re-solve **(2.9).** Usually, the trust region is a ball defined by $\|p\|_2 \leq \Delta$ where the scalar $\Delta > 0$ is called the trust-region radius. Elliptical and box-shaped trust regions may also be used.The model m_k in **(2.9)** is usually defined to be a quadratic function of the form

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p \qquad \textbf{(2.10)}$$

where f_k, ∇f_k, and B_k are a scalar, vector, and matrix, respectively.

As the notation indicates, f_k and ∇f_k are chosen to be the function and gradient values at the point x_k , so that m_k and f are in agreement to first order at the current iterate x_k . The matrix B_k is either the Hessian $\nabla^2 f_k$ or some approximation to it.

Example 2.1:Suppose that the objective function is given by

$f(x) = 10(x_2 - x^2{}_1)^2 + (1 - x_1)^2$. at the point $x_k = (0, 1)$

It's gradient and Hessian are

$$\nabla f_k = \begin{bmatrix} -2 \\ 20 \end{bmatrix}, \quad \nabla^2 f_k = \begin{bmatrix} -38 & 0 \\ 0 & 20 \end{bmatrix}$$

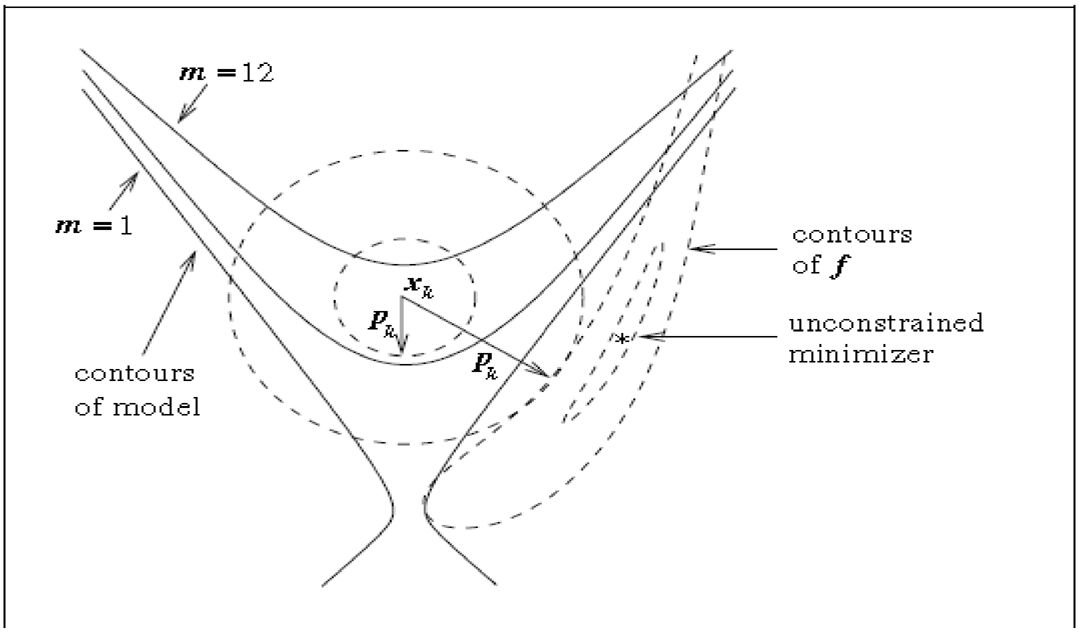


Figure 2.3 Two Possible Trust Regions (Circles) and their Corresponding Steps p_k.

The solid lines are contours of the model function m_k .The contour lines of the quadratic model **(2.10)** with $B_k = \nabla^2 f_k$ are depicted in Figure 2.3,which also illustrates the contours of the objective function f and the trust region. We have indicated contour lines where the model m_k has values 1 and 12. Note from Figure 2.3 that each time we decrease the size of the trust region after failure of a candidate iterate,the step from x_k to the new candidate will be shorter, and it usually points in a different direction from the previous candidate. The trust-region strategy differs in this respect from line search, which stays with a single search direction. In a sense, the line search and trust-region approaches differ in the order in which they choose the direction and distance of the move to the next iterate. Line search starts by fixing the direction p_k and then identifying an appropriate distance, namely the step length α_k. In trust region, we first choose a maximum distance—the trust-region radius Δ_k—and then seek a direction and step that attain the best improvement possible subject to this distance constraint. If this step proves to be unsatisfactory, we reduce the distance measure Δ_k and try again.

2.4 Line Search Methods:

Each iteration of a line search method computes a search direction p_k and then decides how far to move along that direction. The iteration is given by

$$x_{k+1} = x_k + \alpha_k p_k \qquad \textbf{(2.11)}$$

where the positive scalar α_k is called the step length. The success of a line search method depends on effective choices of both the direction p_k and the step length α_k .Most line search algorithms require p_k to be a descent direction one for which $p_k^T \nabla f_k < 0$ because this property guarantees that the function f can be reduced along this direction. Moreover, the search direction often has the form

$$p_k = -B_k^{-1} \nabla f_k \qquad \textbf{(2.12)}$$

Where B_k is a symmetric and nonsingular matrix. In the steepest descent method, B_k is simply the identity matrix , while in Newton's method, B_k is the exact Hessian $\nabla^2 f_k$. In quasi-Newton methods, B_k is an approximation to the Hessian that is updated at every iteration by means of a low-rank formula. When p_k is defined by **(2.12)** and B_k is positive definite, we have

$$p_k^T \nabla f_k = -\nabla f_k^T B_k^{-1} \nabla f_k < 0 \qquad \textbf{(2.13)}$$

and therefore p_k is a descent direction.In this chapter, we discuss how to choose α_k and p_k to promote convergence from remote starting points. Since the pure Newton iteration is not guaranteed to produce descent directions when the current iterate is not close to a solution. We now give careful consideration to the choice of the step-length parameter α_k .

2.5 Step Length:

In computing the step length α_k , we face a tradeoff. We would like to choose α_k to give a substantial reduction of f , but at the same time we do not want to spend too much time making the choice. The ideal choice would be the global minimizer of the univariate function $\emptyset(\cdot)$ defined by

$$\emptyset(\alpha) = f(x_k + \alpha p_k), \ \alpha > 0 \qquad \textbf{(2.14)}$$

But in general, it is too expensive to identify this value **(see Figure 2.4).** To find even a local minimizer of $\emptyset$ to moderate precision generally requires too many evaluations of the objective function f and possibly the gradient . More practical strategies perform an inexact line search to identify a step length that achieves adequate reductions in f at minimal cost.Typical line search algorithms try out a sequence of candidate values for α, stopping to accept one of these values when certain conditions are satisfied. The line search is done in two stages: A bracketing phase finds an interval containing desirable step lengths, and a bisectionor interpolation phase computes a good step length within this interval. Sophisticated linesearch algorithms can be quite complicated.

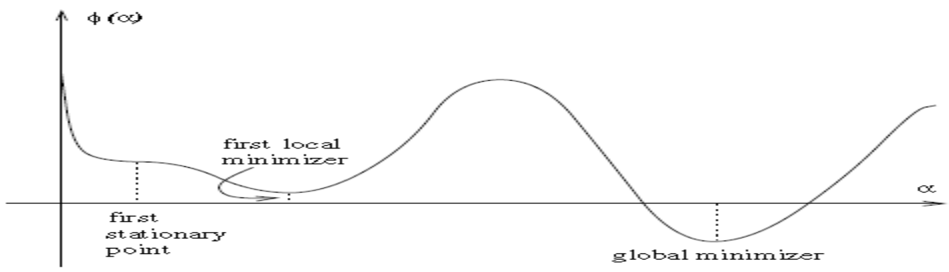


Figure 2.4 The Ideal Step Length is the Global Minimizer.

We now discuss various termination conditions for line search algorithms and show that effective step lengths need not lie near minimizers of the univariate function $\emptyset(\alpha)$ defined in (2.14).A simple condition we could impose on α_k is to require a reduction in f , that is, $f\ (x_k + \alpha_k p_k) < f\ (x_k\ )$. That this requirement is not enough to produce convergence to x^* is illustrated in Figure 2.5, for which the minimum function value is $f^* = -1$, but a sequence of iterates $\{x_k\}$ for which $f(x_k) = \frac{5}{k}, k = 0,1,\ldots$ yields a decrease at each iteration but has a limiting function value of zero. The insufficient reduction in f at each step causes it to fail to converge to the minimizer of this convex function. To avoid this behavior we need to enforce a sufficient decrease condition, a concept we discuss next.

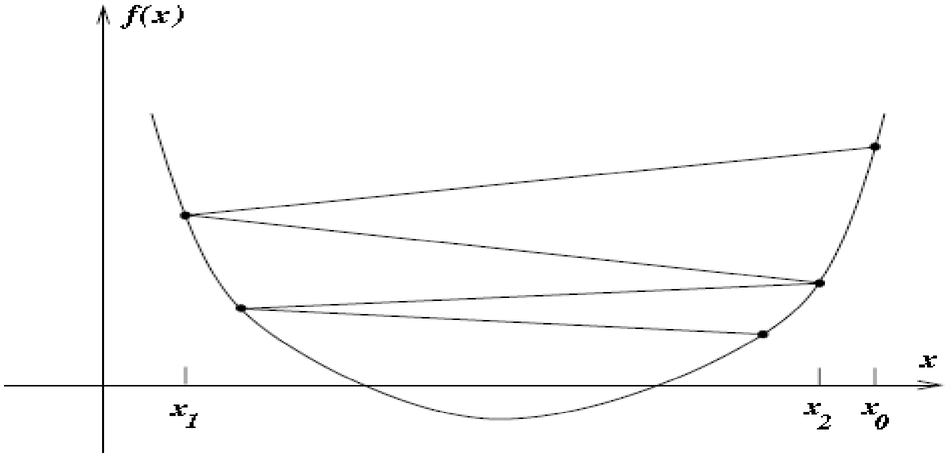


Figure 2.5 Insufficient Reduction in f.

The Wolfe Conditions:

A popular inexact line search condition stipulates that α_k should first of all give sufficient decrease in the objective function f , as measured by the following inequality:

$$f(x_k + \alpha p_k) \le f(x_k) + c_1 \alpha \nabla {f_k}^T p_k \qquad \textbf{(2.15)}$$

for some constant $c_1 \in (0, 1)$. In other words, the reduction in f should be proportional to both the step length α_k and the directional derivative $\nabla f_k^T p_k$. Inequality **(2.15)** is sometimes called the Armijo condition.The sufficient decrease condition is illustrated in Figure 2.6. The right-hand-side of **(2.15)**, which is a linear function, can be denoted by $l(\alpha)$. The function $l(\cdot)$ has negative slope $c_1 \nabla f_k^T p_k$, but because $c_1 \in (0, 1)$, it lies above the graph of $\emptyset$ for small positive values of α. The sufficient decrease condition states that α is acceptable only if $\emptyset(\alpha) \le l(\alpha)$. The intervals on which this condition is satisfied are shown in Figure 2.6 In practice, c_1 is chosen to be quite small, say $c_1 = 10^{-4}$.The sufficient decrease condition is not enough by itself to ensure that the algorithm makes reasonable progress because, as we see from Figure 2.6, it is satisfied for all sufficiently small values of α. To rule out unacceptably short steps we introduce a second requirement, called the curvature condition, which requires α_k to satisfy

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k \qquad \textbf{(2.16)}$$

for some constant $c_2 \in (c_1, 1)$, where c_1 is the constant from (2.15). Note that the left-hand side is simply the derivative $\emptyset'(\alpha_k)$, so the curvature condition ensures that the slope of $\emptyset$ at α_k is greater than c_2 times the initial slope $\emptyset'(0)$. This makes sense because if the slope $\emptyset'(\alpha)$

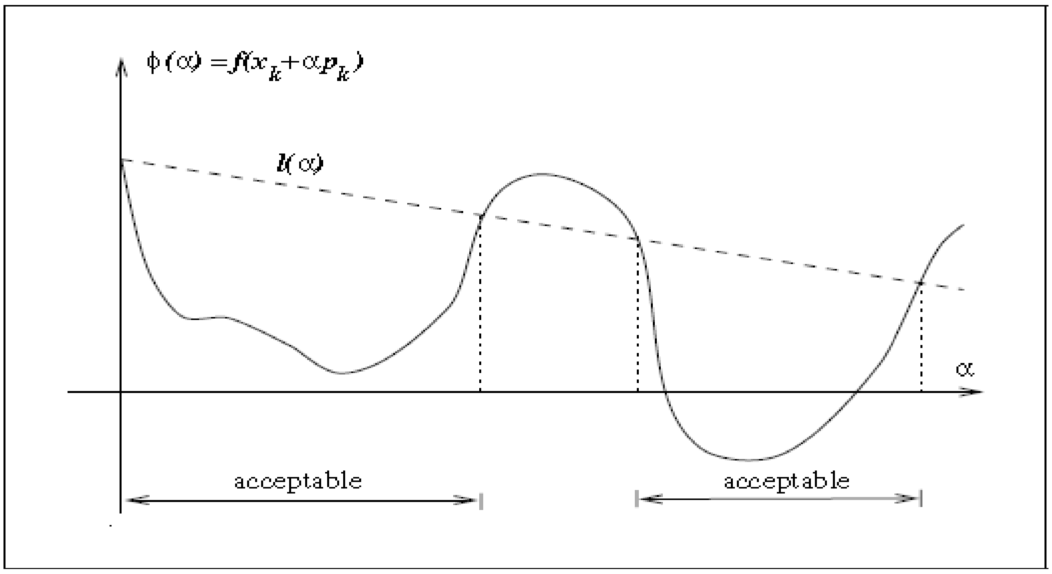


Figure 2.6 Sufficient Decrease Condition.

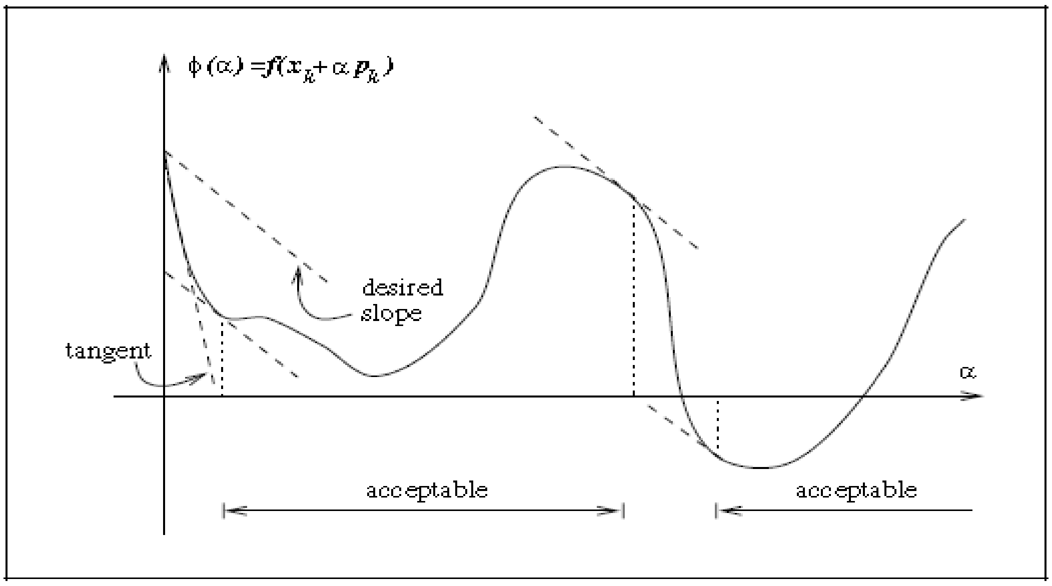


Figure 2.7 The Curvature Condition.

is strongly negative, we have an indication that we can reduce f significantly by moving further along the chosen direction. On the other hand, if $\emptyset'(\alpha_k)$ is only slightly negative or even positive, it is a sign that we cannot expect much more decrease in f in this direction, so it makes sense to terminate the line search. The curvature condition is illustrated in Figure 2.7. Typical values of c_2 are 0.9 when the search direction p_k is chosen by a Newton or quasi-Newton method, and 0.1 when p_k is obtained from a nonlinear conjugate gradient method. The sufficient decrease and curvature conditions are known collectively as the Wolfe conditions.We illustrate them in Figure 2.8 and restate them here for future reference:

$$f\ (x_k + \alpha_k p_k) \le f\ (x_k) + c_1 \alpha_k \nabla f_k^T p_k \qquad \textbf{(2.17a)}$$

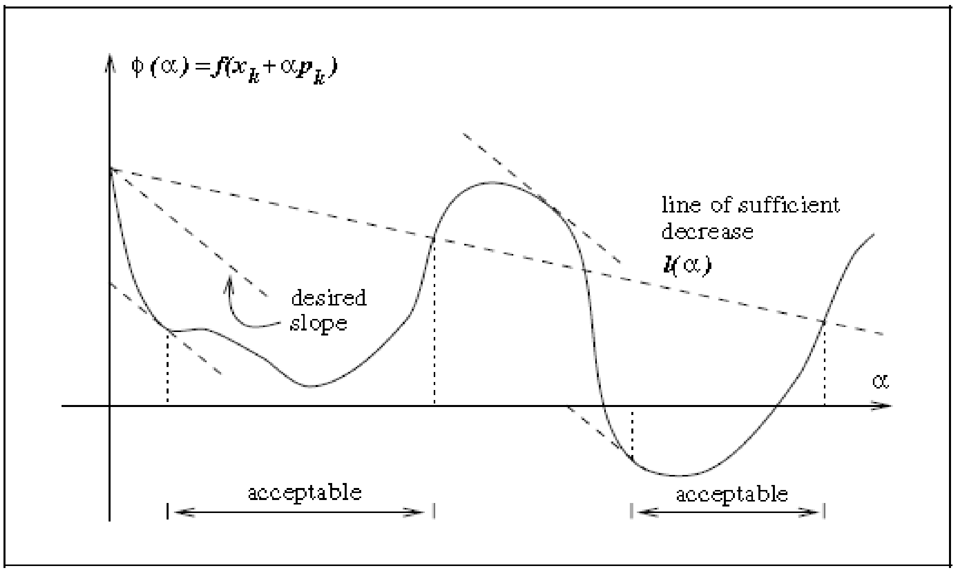
$$\nabla f\ (x_k + \alpha_k p_k)^T p_k \ge c_2 \nabla f_k^T p_k \qquad \textbf{(2.17b)}$$

With $0 < c_1 < c_2 < 1$.A step length may satisfy the Wolfe conditions without being particularly close to a minimizer of $\emptyset$, as we show in Figure2.8.We can, however, modify the curvature condition to force $\alpha_k$to lie in at least a broad neighborhood of a local minimizer or stationary point of $\emptyset$. The strong Wolfe conditions require $\alpha_k$to satisfy

$$f\ (x_k + \alpha_k p_k) \le f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \qquad \textbf{(2.18a)}$$

$$|\nabla f\ (x_k + \alpha_k p_k)^T p_k| \le c_2 \left|\nabla f_k^T p_k\right| \qquad \textbf{(2.18b)}$$

With $0 < c_1 < c_2 < 1$. The only difference with the Wolfe conditions is that we no longer allow the derivative $\emptyset'(\alpha_k\ )$to be too positive. Hence, we exclude points that are far from stationary points of $\emptyset$.



$\phi(\alpha) = f(x_k + \alpha p_k)$
line of sufficient decrease
$l(\alpha)$
desired slope
α
acceptable
acceptable

Figure 2.8 Step Lengths Satisfying the wolfe Conditions.

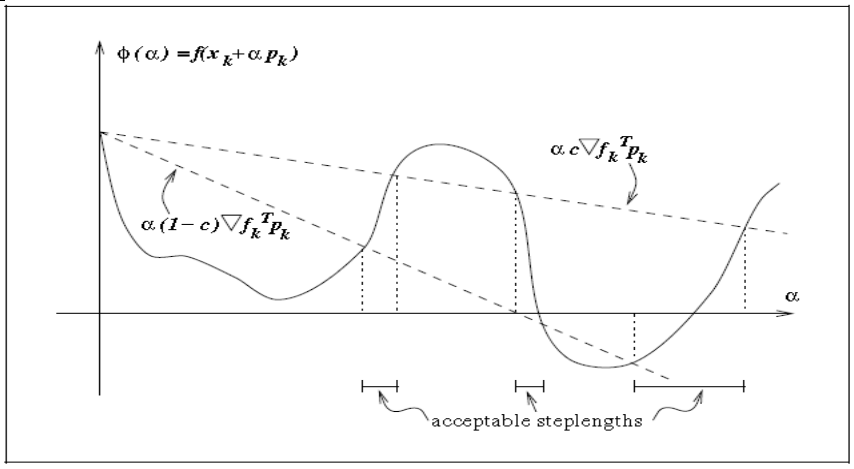
It is not difficult to prove that there exist step lengths that satisfy the Wolfe conditions for every function f that is smooth and bounded below.

The Goldstein Conditions:

Like the Wolfe conditions, the Goldstein conditions ensure that the step length α achieves sufficient decrease but is not too short. The Goldstein conditions can also be stated as a pair of inequalities, in the following way:

$$f(x_k) + (1-c)\alpha_k \nabla f_k^T p_k \le f\ (x_k + \alpha_k p_k) \le f(x_k) + c\alpha_k \nabla f_k^T p_k \qquad \textbf{(2.19)}$$

with $0 < c < 1/2$. The second inequality is the sufficient decrease condition **(2.15)**, where as the first inequality is introduced to control the step length from below; see **Figure 2.8** A disadvantage of the Goldstein conditions vis-`a-vis the Wolfe conditions is that the first inequality in **(2.19)** may exclude all minimizers of $\emptyset$. However, the Goldstein and Wolfe conditions have much in common, and their convergence theories are quite similar. The Goldstein conditions are often used in Newton-type methods but are not well suited for Quasi-Newton methods that maintain a positive definite Hessian approximation.



$\phi(\alpha) = f(x_k + \alpha p_k)$
$\alpha c \nabla f_k^T p_k$
$\alpha(1-c)\nabla f_k^T p_k$
α
acceptable steplengths

Figure 2.9 The Goldstein conditions.

Sufficient Decrease and Backtracking:

We have mentioned that the sufficient decrease condition (2.17a) alone is not sufficient to ensure that the algorithm makes reasonable progress along the given search direction. However, if the line search algorithm chooses its candidate step lengths appropriately, by using a so-called backtracking approach, we can dispense with the extra condition (2.17b) and use just the sufficient decrease condition to terminate the line search procedure. In its most basic form, backtracking proceeds as follows.

Algorithm 2.1 (Backtracking Line Search).

```
Choose $\bar{\alpha} > 0, \rho \in (0,1), c \in (0,1)$;
Set $\alpha \leftarrow \bar{\alpha}$;
repeat until $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha \nabla f_k^T p_k$
    $\alpha \leftarrow \rho\alpha$;
end (repeat)
Terminate with $\alpha_k = \alpha$.
```

In this procedure, the initial step length $\bar{\alpha}$ is chosen to be 1 in Newton and quasi-Newton methods, but can have different values in other algorithms such as steepest descent or conjugate gradient. An acceptable step length will be found after a finite number of trials, because α_k will eventually become small enough that the sufficient decrease condition holds (see Figure 2.6). In practice, the contraction factor ρ is often allowed to vary at each iteration of the line search.

We need ensure only that at each iteration we have $\rho \in [\rho_{lo}, \rho_{hi}]$, for some fixed constants $0 < \rho_{lo} < \rho_{hi} < 1$ .The backtracking approach ensures either that the selected step length α_k is some fixed value (the initial choice $\bar{\alpha}$), or else that it is short enough to satisfy the sufficient decrease condition

but not too short. The latter claim holds because the accepted value α_k is within a factor ρ of the previous trial value, α_k/ρ, which was rejected for violating the sufficient decrease condition, that is, for being too long.
This simple and popular strategy for terminating a line search is well suited for Newton methods but is less appropriate for Quasi-Newton and conjugate gradient methods.

2.6 Convexity:

There is one important case where global solutions can be found, the case where the objective function is a convex function and the feasible region is a convex set. Let us first talk about the feasible region. A set S is convex if, for any elements x and y of S,
$\alpha x + (1 - \alpha)y \in S \text{ for all } 0 \leq \alpha \leq 1$.
In other words, if x and y are in S, then the line segment connecting x and y is also in S.Examples of convex and nonconvex sets are given in Figure 2.9. More generally, every set defined by a system of linear constraints is a convex set;.A function f is convex on a convex set S if it satisfies

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y)$$

for all $0 \leq \alpha \leq 1$ and for all $x, y \in S$. This definition says that the line segment connecting the points $(x, f(x))$ and $(y, f(y))$ lies on or above the graph of the function; see Figure2.5. Intuitively, the graph of the function is bowl shaped.
Analogously, a function is concave on S if it satisfies

$$f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$$

For all $0 \leq \alpha \leq 1$ and for all $x, y \in S$.

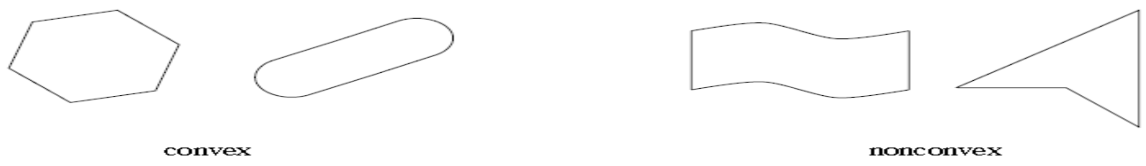


Figure 2.10. Convex and Nonconvex Sets.

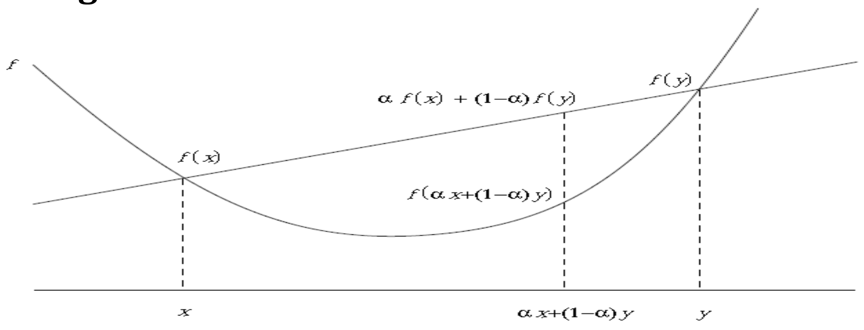


Figure 2.11. Convex function.

Linear functions are both convex and concave.
We say that a function is strictly convex if

$f(\alpha x+(1-\alpha)y) < \alpha f(x)+(1-\alpha)f(y)$ for all $x \neq y$ and $0<\alpha<1$ where $x, y \in S$.

Theorem 2.6:

Let x^* be a local minimizer of a convex optimization problem. Then $x^*$is also a global minimizer. If the objective function is strictly convex, then $x^*$is the unique global minimizer.

Proof.

The proof is by contradiction. Let x^* be a local minimizer and suppose, by contradiction,that it is not a global minimizer. Then there exists some point $y \in S$ satisfying

$f(y) < f(x^*). If$ $0<\alpha<1$,then

$f\ (\alpha x^*+(1-\alpha)y) \leq \alpha f\ (x^*)+(1-\alpha)f(y) < \alpha f(x^*)+(1-\alpha)f(x^*) = f(x^*)$.

This shows that there are points arbitrarily close to x^* (i.e., when α is arbitrarily close to 1) whose function values are strictly less than $f\ (x^*)$. These points are in S because S is convex. This contradicts the definition of a local minimizer. Hence a point such as y cannot exist, and x^* must be a global minimizer.

If the objective function is strictly convex, then a similar argument can be used to show that x^* is the unique global minimize.

For general problems it may be as difficult to determine if the function f and the region S are convex as it is to find a global solution, so this result is not always useful.However, there are important practical problems, such as linear programs, where convexity can be guaranteed [1,12].

2.7 Derivatives and Convexity :

If a one-dimensional function f has two continuous derivatives, then an alternative definition of convexity can be given that is often easier to check. Such a function is convex if and only if $f''(x) \geq 0$ for all $x \in S$;

For example, the function $f(x) = x^4$ is convex on the entire real line because $f(x) = 12x^2 \geq 0$ for all x. The function $f\ (x) = sinx$ is neither convex nor concave on the real line because $f(x) = -sinx$ can be both positive and negative. In the multidimensional case the Hessian matrix of second derivatives must be positive semidefinite; that is, at every point $x \in S$.

$y^T \nabla^2 f(x) y \geq 0$ for all y ;

Notice that the vector y is not restricted to lie in the set S. The quadratic function $f(x_1, x_2) = 4{x_1}^2 + 12x_1x_2 + 9{x^2}_2$

is convex over any subset of R^2 since

$$y^T \nabla^2 f(x) y = (y_1, y_2)\begin{bmatrix} 8 & 12 \\ 12 & 18 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = 8{y^2}_1 + 24y_1y_2 + 18{y^2}_2$$

$= 2(2y_1 + 3y_2)^2 \geq 0.$

Alternatively, it would have been possible to show that the eigenvalues of the Hessian matrix were all greater than or equal to zero. In the one-dimensional case, if a function satisfies
$f''(x) \geq 0$ for all $x \in S$;
Then it is strictly convex on S. In the multidimensional case, if the Hessian matrix $\nabla^2 f(x)$ is positive definite for all $x \in S$, then the function is strictly convex on S. This is not an "if and only if " condition, since the Hessian of a strictly convex function need not be positive definite every where.
Now we consider another characterization of convexity that can be applied to functions that have one continuous derivative. In this case a function f is convex over a convex set S if and only if it satisfies

$$f(y) \geq f(x) + \nabla f(x)^T (y - x)$$

for all $x, y \in S$. This property states that the function is on or above any of its tangents. (See Figure 2.11) To prove this property, note that if f is convex, then for any x and y in S and for any $0 < \alpha \leq 1$,
$f(\alpha y + (1-\alpha)x) \leq \alpha f(y) + (1-\alpha) f(x)$,
so that
$\frac{f(x + \alpha(y-x)) - f(x)}{\alpha} \leq f(y) - f(x)$.
If we let α approach 0 from above, we can conclude that
$f(y) \geq f(x) + \nabla f(x) T (y - x)$.

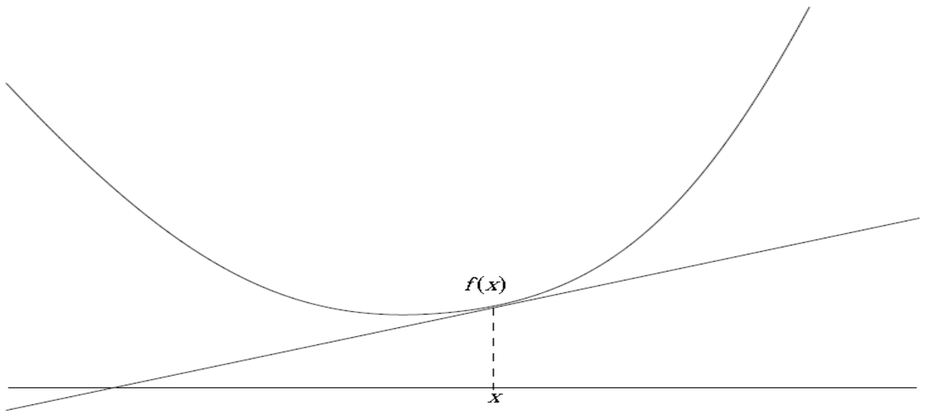


Figure 2.12. Convex Function with Continuous First Derivative.
Conversely, suppose that the function f satisfies
$f(y) \geq f(x) + \nabla f(x) T (y - x)$ for all x and y in S. $Let\ t = \alpha x + (1-\alpha) y$.
Then t is also in the set S, so

$$f(x) \geq f(t) + \nabla f(t)^T (x - t)$$

and

$$f(y) \geq f(t) + \nabla f(t)^T (y - t).$$

Multiplying the two inequalities by α and $1 - \alpha$, respectively, and then adding yields the desired result[1,12].

2.8 Taylor Series:

The Taylor series is a tool for approximating a function f near a specified point x_0. The approximation obtained is a polynomial, i.e., a function that is easy to manipulate. The Taylor series is a general tool it can be applied when ever the function has derivatives and it has many uses:

- It allows you to estimate the value of the function near the given point (when the function is difficult to evaluate directly).
- The derivatives and integral of the approximation can be used to estimate the derivatives and integral of the original function.
- It is used to derive many algorithms for finding zeroes of functions (see below), for minimizing functions, etc.

Since many problems are difficult to solve exactly, and an approximate solution is often adequate (the data for the problem may be in accurate), the Taylor series is widely used, both theoretically and practically. Even if the data are exact, an approximate solution may be adequate, and in any case it is all we can hope for under most circumstances. How does it work? We first consider the case of a one-dimensional function f with n continuous derivatives. Let x_0 be a specified point (say $x_0 = 17.5$ or $x_0 = 0$). Then the nth order Taylor series approximation is

$$f(x_0+p) = f(x_0) + pf'(x_0) + \frac{1}{2}p^2 f''(x_0) + \frac{1}{3!}p^3 f'''(x_0) + \cdots + \frac{1}{n!}p^n f^{(n)}(x_0) \qquad \textbf{(2.20)}$$

Here $f^{(n)}(x_0)$ is the nth derivative of f at the point x_0 ,and $n! = n(n-1)(n-2)\cdots 3 \cdot 2 \cdot 1$. In this formula, p is a variable; we will decide later what values p will take. The approximation will normally only be accurate for small values of p [1,13].

Example 2.2: (Taylor Series). Let $f(x) = \sqrt{x}$ and let $x_0 = 1$. Then

$$f(x_0) = \sqrt{x_0} = \sqrt{1} = 1$$
$$f'(x_0) = \frac{1}{2}x_0^{-\frac{1}{2}} = \frac{1}{2}$$
$$f''(x_0) = -\frac{1}{4}x_0^{-\frac{3}{2}} = -\frac{1}{4}$$
$$f'''(x_0) = \frac{3}{8}x_0^{-\frac{5}{2}} = \frac{3}{8}$$
$$\vdots$$

Hence, substituting into the formula for the Taylor series,

$$f(x_0+p) \approx f(x_0) + pf'(x_0) + \frac{1}{2}p^2 f''(x_0) + \frac{1}{3!}p^3 f'''(x_0) + \cdots + \frac{1}{n!}p^n f^{(n)}(x_0)$$
$$= 1 + \frac{1}{2}p + \frac{1}{2}p^2\left(-\frac{1}{4}\right) + \frac{1}{6}p^3\left(\frac{3}{8}\right)$$

How do we use this? Suppose we want to approximate $f(1.6)$. Then $x_0 + p =$
$1 + p = 1.6$, and so $p = 0.6$:

$$\sqrt{1.6} = \sqrt{1 + 0.6} \approx 1 + \frac{1}{2}(0.6) + \frac{1}{2}(0.6)^2\left(-\frac{1}{4}\right) + \frac{1}{6}(0.6)^3(\frac{3}{8}) \approx 1.2685$$

The true value is 1.264911 . .. ; the approximation is accurate to three digits. The first two terms of the Taylor series give us the formula for the tangent line for the function f at the point x_0. We commonly define the tangent line in terms of a general point x, and not in terms of p. Since $x_0 + p = x$, we can rearrange to get $p = x - x_0$. Substitute this into the first two terms of the series to get the tangent line:
$y = f(x_0) + (x - x_0)f'(x_0)$.

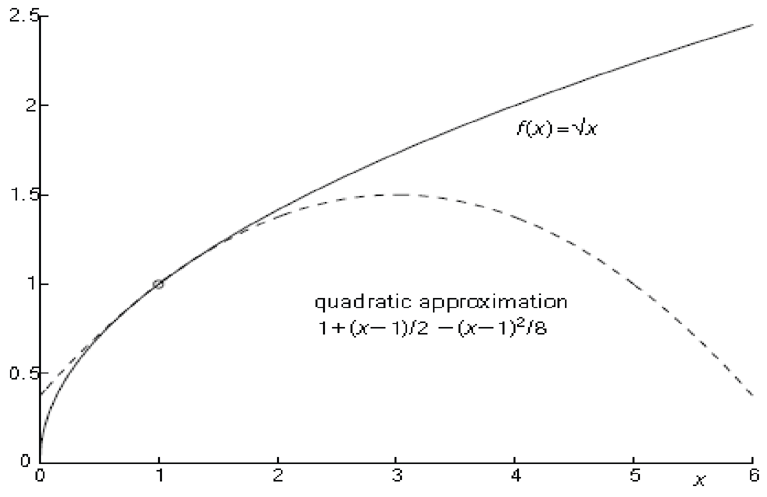


Figure 2.13. Taylor Series Approximation.

For the example above we get
$y = 1 + (x - 1)\frac{1}{2}$ or $y = \frac{1}{2}(x + 1)$.

The first three terms of the Taylor series give a quadratic approximation to the function f at the point x_0. This is illustrated in Figure 2.13. So far we have only considered a Taylor series for a function of one variable. The Taylor series can also be derived for real-valued functions of many variables. If we use matrix and vector notation, then there is an obvious analogy between the two cases:

1-variable: $f(x_0 + p) = f(x_0) + pf'(x_0) + \frac{1}{2}p^2 f''(x_0)$

n-variables: $f(x_0 + p) = f(x_0) + p^T \nabla f(x_0) + \frac{1}{2}p^T \nabla^2 f(x_0)p + \cdots$.

In the second line above x_0 and p are both vectors. The notation $\nabla f(x_0)$ refers to the gradient of the function f at the point $x = x_0$. The notation $\nabla^2 f(x_0)$ represents the Hessian of f at the point $x = x_0$.

Example 2.3: Consider the function

$f(x_1,x_2) = {x_1}^3 + 5{x_1}^2x_2 + 7x_1{x^2}_2 + 2{x^3}_2$ at the point $x_0 = (-2,3)^T$

The gradient of this function is

$$\nabla f(x) = \begin{bmatrix} 3{x_1}^2 + 10x_1x_2 + 7{x^2}_2 \\ 5{x_1}^2 + 14x_1x_2 + 6{x_2}^2 \end{bmatrix}$$

and the Hessian matrix is

$$\nabla^2 f = \begin{bmatrix} 6x_1 + 10x_2 & 10x_1 + 14x_2 \\ 10x_1 + 14x_2 & 14x_1 + 12x_2 \end{bmatrix}$$

at the point $x_0 = (-2,3)^T$ these become

$$\nabla f(x_0) = \begin{bmatrix} 15 \\ -10 \end{bmatrix}, \nabla^2 f(x_0) = \begin{bmatrix} 18 & 22 \\ 22 & 8 \end{bmatrix}$$

If $p = (p_1,p_2)^{\mathrm{T}} = (0.1, 0.2)^T$, then

$$\begin{aligned} f(-1.9, 3.2) &= f(-2 + 0.1, 3 + 0.2) \\ &= f(x_0 + p) \\ &\approx f(x_0) + p^T \nabla f(x_0) + \frac{1}{2} p^T \nabla^2 f(x_0) p \\ &= -20 + (\,0.1\ \ 0.2\,) \begin{bmatrix} 15 \\ -10 \end{bmatrix} + \frac{1}{2} (\,0.1\ \ 0.2\,) \begin{bmatrix} 18 & 22 \\ 22 & 8 \end{bmatrix} \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} \\ &= -20 - 0.5 + 0.69 = -19.81. \end{aligned}$$

The true value is $f(-1.9, 3.2) = -19.755$, so the approximation is accurate to three digits.

The Taylor series for multidimensional problems can also be derived using summations rather than matrix-vector notation:

$$f(x_0 + p) = f(x_0) + \sum_{i=1}^{n} p_i \left.\frac{\partial f(x)}{\partial x_i}\right|_{x=x_0} + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} p_i\, p_j \left.\frac{\partial^2 f(x)}{\partial x_i \partial x_j}\right|_{x=x_0} + \cdots$$

The formula is the same as before; only the notation has changed.

There is an alternate form of the Taylor series that is often used, called the remainder form. If three terms are used it looks like

1-variable: $f(x_0 + p) = f(x_0) + pf'(x_0) + \frac{1}{2} p^2 f''(\xi)$

n-variables: $f(x_0 + p) = f(x_0) + p^T \nabla f(x_0) + \frac{1}{2} p^T \nabla^2 f(\xi) p.$

The point ξ is an unknown point lying between x_0 and $x_0 + p$. In this form the series is exact, but it involves an unknown point, so it cannot be evaluated. This form of the series is often used for theoretical purposes, or to derive bounds on the accuracy of the series.The accuracy of the series can be analyzed by establishing bounds on the final "remainder"term.

If the remainder form of the series is used, but with only two terms, then we obtain

1-variable: $f(x_0 + p) = f(x_0) + pf'(\xi)$

n-variables: $f(x_0 + p) = f(x_0) + p^T \nabla f(\xi).$

This result is known as the mean-value theorem [1,12].

2.9 Rates of Convergence:

Many of the algorithms discussed do not find a solution in a finite number of steps. Instead these algorithms compute a sequence of approximate solutions that we hope get closer and closer to a solution. When discussing such an algorithm, the following two questions are often asked:

- Does it converge?
- How fast does it converge?

It is the second question that is the topic of this section. If an algorithm converges in a finite number of steps, the cost of that algorithm is often measured by counting the number of steps required, or by counting the number of arithmetic operations required. For example, if Gaussian elimination is applied to a system of n linear equations, then it will require about n^3 operations. This cost is referred to as the computational complexity of the algorithm. For many optimization methods, the number of operations or steps required to find an exact solution will be infinite, so some other measure of efficiency must be used. The rate of convergence is one such measure. It describes how quickly the estimates of the solution approach the exact solution.
Let us assume that we have a sequence of points x_k converging to a solution x_*. We define the sequence of errors to be

$$e_k = x_k - x_*$$

Note that

$$\lim_{k\to\infty} e_k = 0$$

We say that the sequence $\{x_k\}$ converges to x_* with rate r and rate constant C if

$$\lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C$$

and $C < \infty$. To understand this idea better, let us look at some examples.
Initially let us assume that we have ideal convergence behavior
$\|e_{k+1}\| = C\|e_k\|^r$ for all k,
so that we can avoid having to deal with limits. When r = 1 this is referred to as linear convergence:
$\|e_{k+1}\| = C\|e_k\|$.
If $0 < C < 1$, then the norm of the error is reduced by a constant factor at every iteration.
If $C > 1$, then the sequence diverges. (What can happen when C = 1?) If we choose $C = 0.1 = 10^{-1}$ and $\|e_0\| = 1$, then the norms of the errors are
$1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7},$

and seven-digit accuracy is obtained in seven iterations, a good result. On the other hand, if $C = 0.99$, then the norms of the errors take on the values

$$1, 0.99, 0.9801, 0.9703, 0.9606, 0.9510, 0.9415, 0.9321, \dots,$$

and it would take about 1600 iterations to reduce the error to 10^{-7}, a less impressive result.

If $r = 1$ and $C = 0$, the convergence is called superlinear. Superlinear convergence includes all cases where $r > 1$ since if

$$\lim_{k\to\infty} \frac{||e_{k+1}||}{||e_k||^r} = C < \infty$$

Then

$$\lim_{k\to\infty} \frac{||e_{k+1}||}{||e_k||} = \lim_{k\to\infty} \frac{||e_{k+1}||}{||e_k||^r} ||e_k||^{r-1} = C \times \lim_{k\to\infty} ||e_k||^{r-1} = 0$$

When $r = 2$, the convergence is called quadratic. As an example, let $r = 2, C = 1$, and $||e_0|| = 10^{-1}$. Then the sequence of error norms is

$10^{-1}, 10^{-2}, 10^{-4}, 10^{-8}$,

and so three iterations are sufficient to achieve seven-digit accuracy.

In this form of quadratic convergence the error is squared at each iteration. Another way of saying this is that the number of correct digits in x_k doubles at every iteration. Of course, if the constant $C = 1$, then this is not an accurate statement, but it gives an intuitive sense of the attractions of a quadratic convergence rate.For optimization algorithms there is one other important case, and that is when $1 < r < 2$.This is another special case of superlinear convergence. This case is important because (a) it is qualitatively similar to quadratic convergence for the precision of common computer calculations, and (b) it can be achieved by algorithms that only compute first derivatives, where as to achieve quadratic convergence it is often necessary to compute second derivatives as well. To get a sense of what this form of superlinear convergence looks like, let $\mathrm{r} = 1.5, \mathrm{C} = 1$, and $||\mathrm{e}_0|| = 10^{-1}$. Then the sequence of error norms is

$1\times10^{-1}, 3\times10^{-2}, 6\times10^{-3}, 4\times10^{-4}, 9\times10^{-6}, 3\times10^{-8}$,

and five iterations are required to achieve single-precision accuracy[1,12].

Chapter 3

Trust-Region Methods

3.1 Introduction:

Line search methods and trust-region methods both generate steps with the help of a quadratic model of the objective function, but they use this model in different ways. Line search methods use it to generate a search direction, and then focus their efforts on finding a suitable step length α along this direction.

Trust-region methods define a region around the current iterate within which they trust the model to be an adequate representation of the objective function, and then choose the step to be the approximate minimizer of the model in this region. In effect, they choose the direction and length of the step simultaneously. If a step is not acceptable, they reduce the size of the region and find a new minimizer. In general, the direction of the step changes whenever the size of the trust region is altered. The size of the trust region is critical to the effectiveness of each step. If the region is too small, the algorithm misses an opportunity to take a substantial step that will move it much closer to the minimizer of the objective function. If too large, the minimizer of the model may be far from the minimizer of the objective function in the region, so we may have to reduce the size of the region and try again. In practical algorithms, we choose the size of the region according to the performance of the algorithm during previous iterations. If the model is consistently reliable, producing good steps and accurately predicting the behavior of the objective function along these steps, the size of the trust region may be increased to allow longer, more ambitious, steps to be taken. A failed step is an indication that our model is an inadequate representation of the objective function over the current trust region. After such a step, we reduce the size of the region and try again. The trust-region approach on a function f of two variables in which the current point x_k and the minimize x^* lie at opposite ends of a curved valley. The quadratic model function m_k, whose elliptical contours are shown as dashed lines, is constructed from function and derivative information at x_k and possibly also on information accumulated from previous iterations and steps. A line search method based on this model searches along the step to the minimizer of m_k (shown), but this direction will yield at most a small reduction in f, even if the optimal step length is used. The trust-region method steps to the minimizer of m_k within the dotted circle (shown), yielding a more significant reduction in f and better progress toward the solution. In this chapter, we will assume that the model function m_k that is used at each iterate m_k is quadratic. Moreover, m_k is based on the Taylor-series expansion of f around x_k, which is

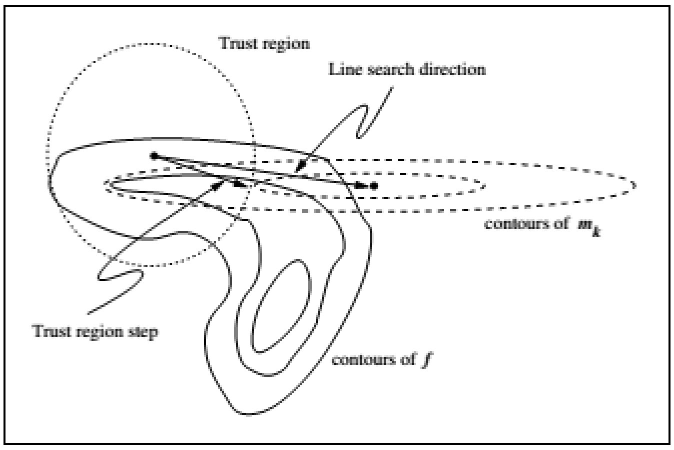


Figure 3.1 Trust-region and line search steps.

$$f(x_k + p) = f_k + g_k^T p + \frac{1}{2} p^T \nabla^2 f(x_k + tp) p \qquad \textbf{(3.1)}$$

where$f_k = f(x_k)$, and $g_k = \nabla f(x_k)$ and t is some scalar in the interval (0,1). By using an approximation B_k to the Hessian in the second-order term, m_k is defined as follows

$$m_k(p) = f_k + g_k^T p + \frac{1}{2} p^T B_k p \qquad \textbf{(3.2)}$$

Where B_k is some symmetric matrix. The difference between $m_k(p)$ and $f(x_k + p)$is $o\|p\|^2$, which is small when p is small.

When B_k is equal to the true Hessian $\nabla^2 f(x_k)$, the approximation error in the model function$m_k$is $o\|p\|^3$, so this model is especially accurate when $\|p\|$ is small. This choice $B_k = \nabla^2 f(x_k)$ leads to the trust-region Newton method. We emphasize the generality of the trust-region approach by assuming little about B_k except symmetry and uniform boundedness. To obtain each step, we seek a solution of the sub-problem

$$\min_{p \in IR^n} (m_k(p)) = f_k + g_k^T p + \frac{1}{2} p^T B_k p \qquad \textbf{(3.3)}$$

Where $\Delta_k > 0$ is the trust-region radius. In most of our discussions, we define $\|\cdot\|$ to be the Euclidean norm, so that the solution p_k^* of **(3.3)** is the minimizer of m_k in the ball of radius Δ_k. Thus, the trust-region approach requires us to solve a sequence of sub-problems **(3.3)** in which the objective function and constraint (which can be written as $p^T p \leq \Delta_k^2$) are both quadratic. When B_k is positive definite and $\|B_k^{-1} g_k\| \leq \Delta_k$, the solution of **(3.3)** is easy to identify it is simply the unconstrained minimum $p_k^B = -B_k^{-1} g_k$ of the quadratic $m_k(p)$. In this case, we call p_k^B the full step see [1]. The solution of **(3.3)** is not so obvious in other cases, but it can usually be found without too much computational expense. In any case, as described below, we need only an approximate solution to obtain convergence and good practical behavior .

3.2 Outline of the Trust-Region Approach :

One of the key ingredients in a trust-region algorithm is the strategy for choosing the trust-region radius Δ_k at each iteration. We base this choice on the agreement between the model function m_k and the objective function f at previous iterations. Given a step p_k we define the ratio

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)} \quad \textbf{(3.4)}$$

The numerator is called the actual reduction, and the denominator is the predicted reduction (that is, the reduction in f predicted by the model function). Note that since the step p_k is obtained by minimizing the model m_k over a region that includes $p = 0$, the predicted reduction will always be non-negative. Hence, if ρ_k is negative, the new objective value $f(x_k + p_k)$ is greater than the current value $f(x_k)$, so the step must be rejected. On the other hand, if ρ_k is close to 1, there is good agreement between the model m_k and the function f over this step, so it is safe to expand the trust region for the next iteration. If ρ_k is positive but significantly smaller than 1, we do not alter the trust region, but if it is close to zero or negative, we shrink the trust region by reducing Δ_k at the next iteration see [1,12,41].

The following algorithm describes the process.

```
Algorithm 3.1 (A Model Trust Region Algorithm)
Given $\hat{\Delta} > 0$ , $\Delta_0 \in (0, \hat{\Delta})$ $and$ $\eta \in \left[0, \frac{1}{4}\right]$:
For $k = 0,1,2, \ldots ..$
Obtain $p_k$ by (approximately) solving (3.3);
Obtaine $\rho_k$ by (3.4)
if $\rho_k < \frac{1}{4}$
                    $\Delta_{k+1} = \frac{1}{4}\Delta_k$
else
if $\rho_k > \frac{3}{4}$ $and$ $\|p_k\| = \Delta_k$
                    $\Delta_{k+1} = min(2\Delta_k, \hat{\Delta})$
```

```
else
                    $\Delta_{k+1} = \Delta_k$
If $\rho_k < \eta$
                    $x_{k+1} = x_k + p_k$
else
                    $x_{k+1} = x_k$
end(for)
```

We need to focus on solving the trust-region sub-problem **(3.3).** In discussing this matter, we sometimes drop the iteration subscript k and restate the problem **(2-1)** as follows

$$\min_{p \in IR^n} m(p) \overset{\text{def}}{=} f + g^T p + \frac{1}{2} p^T B p \quad s.t.\ \|p\| \le \Delta \qquad \textbf{(3.5)}$$

A first step to characterizing exact solutions of (3.5) is given by the following theorem (due to [[1]), which shows that the solution $p^*$of **(3.5)** satisfies

$$(B + \lambda I)p^* = -g \qquad \textbf{(3.6)}$$

for some $\lambda \ge 0$.

Theorem 3.1 :

The vector p^* is a global solution of the trust-region problem

$$\min_{p \in IR^n} m(p) = f + g^T p + \frac{1}{2} p^T B p \quad s.t.\ \|p\| \le \Delta, \qquad \textbf{(3.7)}$$

if and only if p^* is feasible and there is a scalar $\lambda \ge 0$ such that the following conditions are satisfied:

$$(B + \lambda I)p^* = -g, \qquad \textbf{(3.8a)}$$

$$\lambda(\Delta - \|p^*\|) = 0, \qquad \textbf{(3.8b)}$$

$$(B + \lambda I) \text{is positive semi-definite.} \qquad \textbf{(3.8c)}$$

We delay the proof of this result until Section **(3.3),** and instead discuss just its key features here with the help of Figure (3.2). The condition **(3.8b)** is a complementarily condition that states that at least one of the nonnegative quantities λ and $(\Delta - \|p^*\|)$ must be zero. Hence, when the solution lies strictly inside the trust region (as it does when $\Delta = \Delta_1$ in Figure (3.2), we must have $\lambda = 0$ and so $Bp^* = -g$ with B positive semi-definite, from **(3.8a) and (3.8c),** respectively. In the other cases $\Delta = \Delta_2$ and $\Delta = \Delta_3$, we have $\|p^*\| = \Delta$, and so λ is all owed to take a positive value. Note from **(3.8a)** that

$$\lambda p^* = -Bp^* - g = -\nabla m(p^*).$$

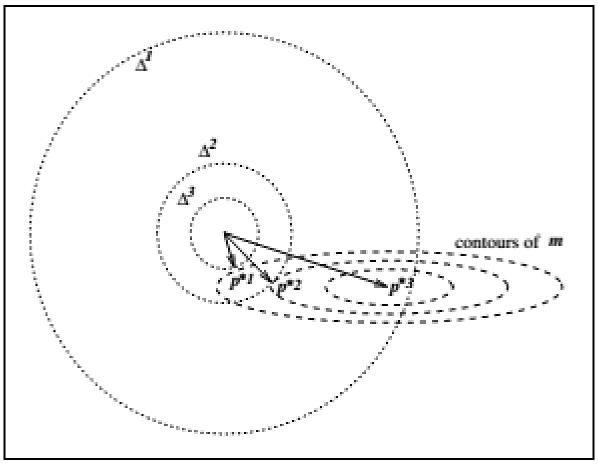


Figure 3.2 Solution of trust-region sub-problem for different radius $\Delta^1, \Delta^2, \Delta^3$

thus, when $\lambda > 0$, the solution p^* is collinear with the negative gradient of m and normal to its contours. These properties can be seen in **Figure (3.2).** In this Section, we describe two strategies for finding approximate solutions of the sub-problem **(3.3),** which achieve at least as much reduction in m_k as the reduction achieved by the so-called iterative method is used to identify the value of λ for which (3.6) is satisfied by the solution of the sub-problem. The second strategy truncated Newton Method.

3.3 Iterative Solution of the Sub-problem :

In this section, we describe a technique that uses the characterization **(3.6)** of the sub-problem solution, applying Newton's method to find the value of λ which matches the given trust-region radius Δ in **(3.5).** We also prove the key result Theorem (3.1) concerning the characterization of solutions of **(3.7).** The characterization of Theorem (3.1) suggests an algorithm for finding the solution p of **(3.7).**Either $\lambda = 0$ satisfies **(3.8a) and (3.8c)** with $||p|| \leq \Delta$, or else we define

$$p(\lambda) = -(B + \lambda I)^{-1} g$$

For λ sufficiently large that $B + \lambda I$ is positive definite and seek a value $\lambda > 0$ such that

$$||p(\lambda)|| = \Delta. \qquad \textbf{(3.9)}$$

This problem is a one-dimensional root-finding problem in the variable λ. To see that a value of λ with all the desired properties exists, we appeal to the eigen-de-composition of B and use it to study the properties of $||p(\lambda)||$.Since$B$ is symmetric, there is an orthogonal matrix Q and a diagonal matrix Λ such that $B = Q\Lambda Q^T$, where

$$\Lambda = diag(\lambda_1, \lambda_2, \ldots, \lambda_n),$$

and$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigen-values of B. Clearly,

$$B + \lambda I = Q(\Lambda + \lambda I)Q^T,$$

and for $\lambda \neq \lambda_j$,we have

$$p(\lambda) = -Q(\Lambda + \lambda I)^{-1}Q^T g = -\sum_{j=1}^{n} \frac{q_j^T g}{\lambda_j + \lambda} q_j, \qquad \textbf{(3.10)}$$

whereq_j denotes the jth column of Q. Therefore, by orthogonality of $q_1, q_2, \ldots, q_n$, we have

$$\|p(\lambda)\|^2 = \sum_{j=1}^{n} \frac{\left(q_j^T g\right)^2}{\left(\lambda_j + \lambda\right)^2}. \qquad \textbf{(3.11)}$$

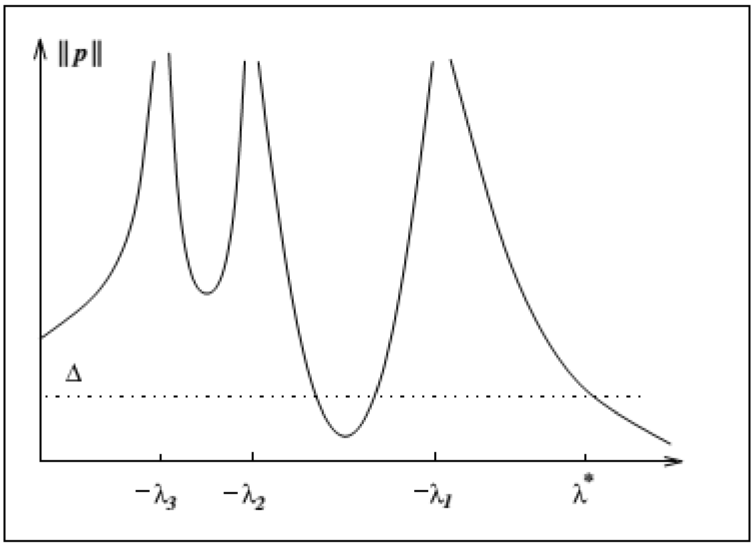


Figure 3.3 $\|p(\lambda)\|$ **as a function of** $\boldsymbol{\lambda}$

This expression tells us a lot about $p(\lambda)$.If$\lambda > -\lambda_1$, we have $\lambda_j + \lambda > 0$ for all $j = 1,2, \ldots, n$, and so $\|p(\lambda)\|$ is a continuous, non-increasing function of λ on the interval $(-\lambda_1, \infty)$. In fact, we have that

$$\lim_{\lambda \to \infty} \|p(\lambda)\| = 0. \qquad \textbf{(3.12)}$$

Moreover, we have when $q_j^T g \neq 0$q that

$$\lim_{\lambda \to -\lambda_j} \|p(\lambda)\| = \infty. \qquad \textbf{(3.13)}$$

Figure (3.3) plots $\|p(\lambda)\|$ against λ in a case in which $q_1^T g, q_2^T g$, and $q_3^T g$ are all nonzero. Note that the properties **(3.12)** and **(3.13)** hold and that

$\|p(\lambda)\|$ is a non-increasing function of λ on $(-\lambda_1, \infty)$. In particular, as is always the case when $q_1^T g \neq 0$, that there is a unique value $\lambda^* \in (-\lambda_1, \infty)$ such that $\|p(\lambda^*)\| = \Delta$. (There may be other, smaller values of λ for which $\|p(\lambda)\| = \Delta$, but these will fail to satisfy **(3.8c)**. We now sketch a procedure for identifying the $\lambda^* \in (-\lambda_1, \infty)$ for which $\|p(\lambda^*\| = \Delta$, which works when $q_1^T g \neq 0$. (We discuss the case of $q_1^T g = 0$ later.) First, note that when B positive definite and $\|B^{-1} g\| \leq \Delta$, the value $\lambda = 0$ satisfies **(3.8)**, so the procedure can be terminated immediately with $\lambda^* = 0$. Otherwise, we could use the root-finding Newton's method to find the value of $\lambda > -\lambda_1$ that solves

$$\emptyset_1(\lambda) = \|p(\lambda)\| - \Delta = 0. \tag{3.14}$$

The disadvantage of this approach can be seen by considering the form of $\|p(\lambda)\|$ when λ is greater than, but close to, $-\lambda_1$.For such λ, we can approximate $\emptyset_1$ by a rational function, as follows

$$\emptyset_1(\lambda) \approx \frac{C_1}{\lambda + \lambda_1} + C_2,$$

where $C_1 > 0$ and C_2 are constants. Clearly this approximation (and hence $\emptyset_1$) is highly nonlinear, so the root-finding Newton's method will be unreliable or slow. Better results will be obtained if we reformulate the problem **(3.14)** so that it is nearly linear near the optimal λ.By defining

$$\emptyset_2(\lambda) = \frac{1}{\Delta} - \frac{1}{\|p(\lambda)\|},$$

It can be shown using **(3.11)** that for λ slightly greater than $-\lambda_1$, we have

$$\emptyset_2(\lambda) \approx \frac{1}{\Delta} - \frac{\lambda + \lambda_1}{C_3}$$

for some $C_3 > 0$. Hence, $\emptyset_2$ is nearly linear near $-\lambda_1$(see Figure (3.4), and the root-finding

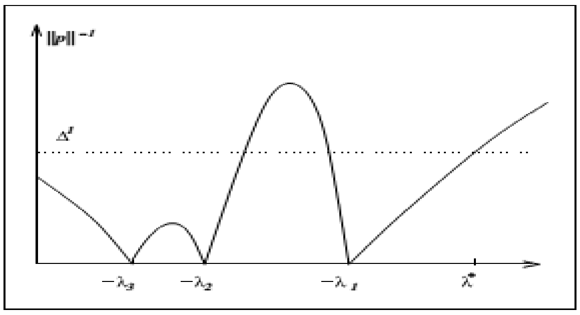


Figure 3.4 $1/\|p(\lambda)\|$ as a function of λ.

Newton's method will perform well, provided that it maintains $\lambda > -\lambda_1$.The root-finding Newton's method applied to $\emptyset_2$ generates a sequence of iterates $\lambda^{(l)}$ by setting

$$\lambda^{(l+1)} = \lambda^{(l)} - \frac{\emptyset_2(\lambda^{(l)})}{\emptyset_2'(\lambda^{(l)})}. \qquad \textbf{(3.15)}$$

After some elementary manipulation, this updating formula can be implemented in the following practical way see [1].

Algorithm (3.2) (Trust Region Sub-problem)

```
$Given\ \lambda^{(0)}, \Delta > 0$:
    $for\ \ell = 0,1,2, \ldots$
        $Factor\ B + \lambda^{(\ell)} I = R^T R$;
        $Solve\ R^T R p_\ell = -g, R^T q_\ell = p_\ell$;
        $Set$
        $\lambda^{(\ell+1)} = \lambda^{(\ell)} + \left(\frac{\|p_\ell\|}{\|q_\ell\|}\right)^2 \left(\frac{\|p_\ell\| - \Delta}{\Delta}\right)$;   (3.16)
    $end\ (for)$.
```

Safeguards must be added to this algorithm to make it practical for instance, when $\lambda^{(l)} < -\lambda_1$the Cholesky factorization

$$B + -\lambda^l I = R^T R$$

will not exist. A slightly enhance version of this algorithm does, however, converge to a solution of **(3.9)** in most cases. The main work in each iteration of this method is, of course, the Cholesky factorization of $B + -\lambda^l I$. Practical versions of this algorithm do not iterate until convergence to the optimal λ is obtained with high accuracy, but are content with an approximate solution that can be obtained in two or three iterations see [71].

Example 3.1:

Consider $f: IR^n \rightarrow IR$ defined by $f(x) = x_1^3 + 3x_1x_2^2$

and let $x = (0,0)^T$ and $p = (1,2)^T$

$$f(x+p) = f(x) + \nabla f(x + \alpha p)^T p$$

$$f(x+p) = (x_1 + p_1)^3 + 3(x_1 + p_1)(x_2 + p_2)^2 \qquad (*)$$

by substituting $x \; and \; p$ in $(*)$

$$f(x+p) = (0+1)^3 + 3(0+1)(0+2)^2 = 1 + 3 \times 4 = 13$$

$$f(x+\alpha p) = (x_1+\alpha p_1)^3 + 3(x_1+\alpha p_1)(x_2+\alpha p_2)^2$$

$$\nabla f(x+\alpha p) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 3(x_1+\alpha p_1)^2 + 3(x_2+\alpha p_2)^2 \\ 6(x_1+\alpha p_1)(x_2+\alpha p_2) \end{bmatrix}$$

$$= \begin{bmatrix} 3(0+\alpha \times 1)^2 + 3(0+\alpha \times 2)^2 \\ 6(0+\alpha \times 1)(0+\alpha \times 2) \end{bmatrix} = \begin{bmatrix} 15\alpha^2 \\ 12\alpha^2 \end{bmatrix}$$

$$\nabla f(x+\alpha p)^T p = \begin{bmatrix} 15\alpha^2 & 12\alpha^2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = [15\alpha^2 + 24\alpha^2] = [39\alpha^2]$$

$$f(x+p) = f(x) + \nabla f(x+p)^T p$$

$$13 = 0 + 39\alpha^2 \Rightarrow \alpha^2 = \frac{13}{39} \Rightarrow \alpha^2 = \frac{1}{3} \Rightarrow \alpha = \frac{1}{\sqrt{3}}$$

Example 3.2:

Find $\nabla f(x)$ and $\nabla^2 f(x)$, If $x = (1,1)^T$

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

Solution

$$f(x) = 100(x_2^2 - 2x_2x_1^2 + x_1^4) + (1 - 2x_1 + x_1^2)$$

$$= 100x_2^2 - 200x_2x_1^2 + 100x_1^4 + 1 - 2x_1 + x_1^2$$

$$100x_1^4 + x_1^2 - 2x_1 + 100x_2^2 - 200x_2x_1^2 + 1$$

$$\frac{\partial f}{\partial x_1} = 400x_1^3 + 2x_1 - 2 - 400x_2x_1$$

$$\frac{\partial f}{\partial x_2} = 200x_2 - 200x_1^2$$

$$\nabla f(x) = \begin{bmatrix} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 400x_1^3 + 2x_1 - 2 - 400x_2x_1 \\ 200x_2 - 200x_1^2 \end{bmatrix} = \begin{bmatrix} 400+2-2-400 \\ 200-200 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\frac{\partial^2 f}{\partial x_1^2} = 1200x_1^2 + 2 - 400x_2$$

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} = -400x_1 \frac{\partial^2 f}{\partial x_2 \partial x_1} = -400x_1$$

$$\frac{\partial^2 f}{\partial x_2^2} = 200$$

$$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 1200x_1^2 + 2 - 400x_2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}$$

$$= \begin{bmatrix} 1200 + 2 - 400 & -400 \\ -400 & 200 \end{bmatrix} = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix}$$

3.4 Truncated Newton Method:

In this section, we study truncated Newton method (modified Newton CG), which is assumed to be the solution of the trust region subproblem(3.3). This algorithm, due to Steihaug [1], is specified below as Algorithm 3.3. A complete algorithm for minimizing f is obtained by using Algorithm 3.3 to generate the step p_k required by Algorithm 3.1, for some choice of tolerance ϵ_k at each iteration. we use d_j to denote the search directions of this modified CG iteration and z_j to denote the sequence of iterates that it generates[1,12].

Algorithm 3.3 (CG-Steihaug)

```
Given tolerance $\epsilon_k > 0$ ;
Set $z_0 = 0, r_0 = \nabla f_k, d_0 = -r_0 = -\nabla f_k$;
If $\|r_0\| < \epsilon_k$
Return $p_k = z_0 = 0$;
For $j = 0,1,2,\ldots..$
If $d_j^T B_k d_j \leq 0$
Find $\tau$ such that $p_k = z_j + \tau d_j$ minimizes $m_k(p_k)$ in **Algorithm 3.1** and satisfies $\|p_k\| \leq \Delta_k$
Return $p_k$;
Set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$ ;
Set $z_{j+1} = z_j + \alpha_j d_j$;
If $\|z_{j+1}\| \geq \Delta_k$
Find $\tau \geq 0$ such that $p_k = z_j + \tau d_j$ satisfies $\|p_k\| = \Delta_k$;
Return $p_k$ ;
```

```
Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
If $\|r_{j+1}\| < \epsilon_k$
Return $p_k = z_{j+1}$;
Set $\beta_{j+1} = r_{j+1}^T r_{j+1}/r_j^T r_j$;
Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
End(for)
```

The initialization of z_0 to zero in Algorithm 3.3 is a crucial feature of the algorithm. Provided $\|\nabla f_k\|_2 \geq \epsilon_k$, Algorithm 3.3 terminates at a point p_k for which $m_k(p_k) \leq m_k(p_k^c)$, that is, when the reduction in model function equals or exceeds that of the Cauchy point. To demonstrate this fact, we consider several cases. First, if $d_0^T B_k d_0 = (\nabla f_k)^T B_k \nabla f_k \leq 0$, then the condition in the first if statement is satisfied, and the algorithm returns the Cauchy point $p = -\Delta_k (\nabla f_k)/\|\nabla f_k\|$. Otherwise, Algorithm 3.3 defines z_1 as follows:

$$z_1 = \alpha_0 d_0 = [r_0^T r_0/d_0^T B_k d_0] d_0 = -\frac{(\nabla f_k)^T \nabla f_k}{(\nabla f_k)^T B_k \nabla f_k} \nabla f_k.$$

If $\|z_1\| < \Delta_k$, then z_1 is exactly the Cauchy point. Subsequent steps of Algorithm 3.3 ensure that the final p_k satisfies $m_k(p_k) \leq m_k(z_1)$.

When $\|z_1\| \geq \Delta_k$, on the other hand, the second if statement is activated, and Algorithm 3.3 terminates at the Cauchy point, proving our claim. This property is important for global convergence: Since each step is at least as good as the Cauchy point in reducing the model m_k , Algorithm 3.3 is globally convergent. Another crucial property of the method is that each iterate z_j is larger in norm than its predecessor. This property is another consequence of the initialization $z_0 = 0$. Its main implication is that it is acceptable to stop iterating as soon as the trust-region boundary is reached, because no further iterates giving a lower value of the model function m_k will lie inside the trust region.

Theorem 3.2.

The sequence of vectors $\{z_j\}$ generated by Algorithm 3.3 satisfies

$$0 = \|z_0\|_2 < \cdots \|z_j\|_2 < \|z_{j+1}\|_2 < \cdots \|p_k\|_2 \leq \Delta_k .$$

PROOF. We first show that the sequences of vectors generated by Algorithm 3.3 satisfy $z_j^T r_j = 0$ for $j \geq 0$ and $z_j^T d_j > 0$ for $j \geq 1$. Algorithm 3.3 computes z_{j+1} recursively in terms of z_j ; but when all the terms of this recursion are written explicitly, we see that

$$z_j = z_0 + \sum_{i=0}^{j-1} \alpha_i d_i = \sum_{i=0}^{j-1} \alpha_i d_i$$

since $z_0 = 0$ .Multiplying by r_j and applying the expanding subspace property of conjugate gradients

$$z_j^T r_j = \sum_{i=0}^{j-1} \alpha_i d_i^T r_j = 0 \qquad \textbf{(3.17)}$$

An induction proof establishes the relation $z_j^T d_j > 0$. By applying the expanding subspace property again, we obtain

$$z_1^T d_1 = (\alpha_0 d_0)^T(-r_1 + \beta_1 d_0) = \alpha_0 \beta_1 d_0^T d_0 > 0.$$

We now make the inductive hypothesis that $z_j^T d_j > 0$ and deduce that $z_{j+1}^T d_{j+1} > 0$. From **(3.17),** we have $z_{j+1}^T r_{j+1} = 0$, and therefore

$$z_{j+1}^T d_{j+1} = z_{j+1}^T(-r_{j+1} + \beta_{j+1} d_j)$$
$$\beta_{j+1}\, z_{j+1}^T d_j = \beta_{j+1}(z_j + \alpha_j d_j)^T d_j$$
$$\beta_{j+1} z_j{}^T d_j + \alpha_j \beta_{j+1} d_j{}^T d_j$$

Because of the inductive hypothesis and positivity of $\beta_{j+1}$and α_j , the last expression is positive. We now prove the theorem. If Algorithm 3.3 terminates because $d_j{}^T B_k d_j \leq 0$ or$\|z_{j+1}\|_2 \geq \Delta_k$, then the final point p_k is chosen to make $\|p_k\|_2 = \Delta_k$, which is the largest possible length. To cover all other possibilities in the algorithm, we must show that $\|z_j\|_2 < \|z_{j+1}\|_2$whenz$_{j+1} = z_j + \alpha_j d_j$ and $j \geq 1$. Observe that

$$\|z_{j+1}\|_2{}^2 = (z_j + \alpha_j d_j)^T(z_j + \alpha_j d_j) = \|z_j\|_2^2 + 2\alpha_j z_j{}^T d_j + \alpha_j{}^2\|d_j\|_2^2$$

It follows from this expression and our intermediate result that $\|z_j\|_2 < \|z_{j+1}\|_2$, so our proof is complete.

Chapter 4

Quasi-Newton Methods

4.1 Introduction :

In the mid 1950s, W.C. Davidon, a physicist working at Argonne National Laboratory, was using the coordinate descent method to perform a long optimization calculation. At that time computers were not very stable, and to Davidon's frustration, the computer system would always crash before the calculation was finished. So Davidon decided to find a way of accelerating the iteration. The algorithm he developed—the first quasi-Newton algorithm—turned out to be one of the most creative ideas in nonlinear optimization. It was soon demonstrated by Fletcher and Powell that the new algorithm was much faster and more reliable than the other existing methods, and this dramatic advance transformed nonlinear optimization overnight.

During the following twenty years, numerous variants were proposed and hundreds of papers were devoted to their study. An interesting historical irony is that Davidon's paper [110] was not accepted for publication; it remained as a technical report for more than thirty years until it appeared in the first issue of the SIAM Journal on Optimization in 1991 [111].

Quasi-Newton methods, like steepest descent, require only the gradient of the objective function to be supplied at each iterate. By measuring the changes in gradients, they construct a model of the objective function that is good enough to produce superliner convergence.

Moreover, since second derivatives are not required, Quasi-Newton methods are sometimes more efficient than Newton's method[1,12].

4.2 The BFGS Method:

The most popular Quasi-Newton algorithm is the BFGS method, named for its discoverers Broyden, Fletcher, Goldfarb, and Shanno.

We begin the derivation by forming the following quadratic model of the objective function at the current iterate x_k :

$$m_k(p) = f_k + \nabla f^T{}_k p + \frac{1}{2} p^T B_k p. \qquad \textbf{(4.1)}$$

Here B_k is an $n \times n$ symmetric positive definite matrix that will be revised or updated at every iteration. Note that the function value and gradient of this model at $p = 0$ match $f_k$and ∇f_k , respectively. The minimize p_k of this convex quadratic model, which we can write explicitly as

$$p_k = -B^{-1}{}_k \nabla f_k \qquad \textbf{(4.2)}$$

is used as the search direction, and the new iterate is

$$x_{k+1} = x_k + \alpha_k p_k \qquad \textbf{(4.3)}$$

where the step length α_k is chosen to satisfy the Wolfe conditions (2.15).

Instead of computing B_k a fresh at every iteration, Davidon proposed to update it in a simple manner to account for the curvature measured during the most recent step. Suppose that we have generated a new iterate x_{k+1} and wish to construct a new quadratic model, of the form

$m_{k+1}(p) = f_{k+1} + \nabla f^T{}_{k+1} p + \frac{1}{2} p^T B_{k+1} p.$

What requirements should we impose on B_{k+1}, based on the knowledge gained during the latest step? One reasonable requirement is that the gradient of m_{k+1} should match the gradient of the objective function f at the latest two iterates x_k and x_{k+1}. Since $\nabla m_{k+1}(0)$ is precisely ∇f_{k+1}, the second of these conditions is satisfied automatically. The first condition can be written mathematically as

$$\nabla m_{k+1}(-\alpha_k p_k) = \nabla f_{k+1} - \alpha_k B_{k+1} p_k = \nabla f_k$$

By rearranging, we obtain

$$B_{k+1} \alpha_k p_k = \nabla f_{k+1} - \nabla f_k. \quad \textbf{(4.4)}$$

To simplify the notation it is useful to define the vectors

$$s_k = x_{k+1} - x_k = \alpha_k p_k, \; y_k = \nabla f_{k+1} - \nabla f_k \quad \textbf{(4.5)}$$

so that **(4.4)** becomes

$$B_{k+1} s_k = y_k \quad \textbf{(4.6)}$$

We refer to this formula as the secant equation.

Given the displacement s_k and the change of gradients y_k, the secant equation requires that the symmetric positive definite matrix B_{k+1} map s_k into y_k. This will be possible only if s_k and y_k satisfy the curvature condition

$$s_k^T y_k > 0 \quad \textbf{(4.7)}$$

as is easily seen by premultiplying **(4.6)** by s_k^T .When f is strongly convex, the inequality **(4.7)** will be satisfied for any two points x_k and x_{k+1} .However, this condition will not always hold for nonconvex functions, and in this case we need to enforce **(4.7)** explicitly, by imposing restrictions on the line search procedure that chooses the step length $\boldsymbol{\alpha}$. In fact, the condition **(4.7)** is guaranteed to hold if we impose the Wolfe **(2.15)** on the line search. To verify this claim, we note from **(4.5**) that $\nabla f_{k+1}^T s_k \geq c_2 \nabla f_k^T s_k$, and therefore

$$y_k^T s_k \geq (c_2 - 1) \nabla f_k^T p_k \quad \textbf{(4.8)}$$

Since $c_2 < 1$ and since p_k is a descent direction, the term on the right is positive, and the curvature condition **(4.7)** holds. When the curvature condition is satisfied, the secant equation **(4.6)** always has a solution B_{k+1}. In fact, it admits an infinite number of solutions, since the $\frac{n(n+1)}{2}$ degrees of freedom in a symmetric positive definite matrix exceed the n conditions imposed by the secant equation. The requirement of positive definiteness imposes n additional inequalities (all principal minors must be positive)but these conditions do not absorb the remaining degrees of freedom. To determine B_{k+1} uniquely, we impose the additional condition that among all symmetric matrices satisfying the secant equation, $B_{k+1}$is, in some sense, closest to the current matrix B_k . In other words, we solve the problem

$$\min_B \|B - B_k\| \qquad \textbf{(4.9a)}$$

$$\text{Subject to } B = B^T, Bs_k = y_k \qquad \textbf{(4.9b)}$$

Where s_k and y_k satisfy **(4.7)** and B_k is symmetric and positive definite. Different matrix norms can be used in **(4.9a)**, and each norm gives rise to a different quasi-Newton method. A norm that allows easy solution of the minimization problem **(4.9)** and gives rise to a scale-invariant optimization method is the weighted Frobenius norm

$$\|A\|_W \equiv \left\|W^{1/2} A W^{1/2}\right\|_F, \qquad \textbf{(4.10)}$$

The weight matrix W can be chosen as Many matrix satisfying the relation $y_k = s_k$ .With this weighting matrix and this norm, the unique solution of **(4.9)** is

$$\textbf{(DFP)} B_{k+1} = \left(I - \rho_k y_k s_k^T\right) B_k \left(I - \rho_k s_k y_k^T\right) + \rho_k y_k y_k^T \qquad \textbf{(4.11)}$$

With

$$\rho_k = \frac{1}{y_k^T s_k} \qquad \textbf{(4.12)}$$

This formula is called the DFP updating formula, since it is the one originally proposed by Davidon in 1959, and subsequently studied, implemented, and popularized by Fletcher and Powell. The inverse of B_k, which we denote by

$$H_k = B_k^{-1}$$

is useful in the implementation of the method, since it allows the search direction **(4.2)** to be calculated by means of a simple matrix–vector multiplication. Using the Sherman–Morrison–Woodbury formula we can derive the following expression for the update of the inverse Hessian approximation H_k that corresponds to the DFP update of $B_k$in **(4.11):**

$$\textbf{DFP} \quad H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} - \frac{s_k s_k^T}{y_k^T s_k} \qquad \textbf{(4.13)}$$

The DFP updating formula is quite effective, but it was soon superseded by the BFGS formula, which is presently considered to be the most effective of all Quasi-Newton Updating formulae. BFGS updating can be derived by making a simple change in the argument that led to **(4.11).** Instead of imposing conditions on the Hessian approximations B_k, we impose similar conditions on their inverses H_k . The updated approximation H_{k+1} must be symmetric and positive definite, and must satisfy the secant equation **(4.6),** now written as

$$H_{k+1} y_k = s_k$$

The condition of closeness to H_k is now specified by the following analogue of **(4.9)**:

$$\min_B \|H - H_k\| \qquad \textbf{(4.14a)}$$
$$\text{Subject to } H = H^T, Hy_k = sy_k \qquad \textbf{(4.14b)}$$

The norm is again the weighted Frobenius norm described above, where the weight matrix W is now any matrix satisfying $s_k = y_k$.

The unique solution H_{k+1} to **(4.14)** is given by

(BFGS) $$H_{k+1} = \left(I - \rho_k s_k y_k^T\right) H_k \left(I - \rho_k y_k s_k^T\right) + \rho_k s_k s_k^T \qquad \textbf{(4.15)}$$

With ρ_k defined by **(4.12).**

Just one issue has to be resolved before we can define a complete BFGS algorithm: How should we choose the initial approximation H_0 ? Unfortunately, there is no magic formula that works well in all cases. We can use specific information about the problem, for instance by setting it to the inverse of an approximate Hessian calculated by finite differences at x_0 . Otherwise, we can simply set it to be the identity matrix, or a multiple of the identity matrix, where the multiple is chosen to reflect the scaling of the variables [1].

Algorithm 4.1 (BFGS Method).

```
Given starting point $x_0$, convergence tolerance $\epsilon > 0$, inverse Hessian
approximation $H_0$;
$k \leftarrow 0$;
While $\|\nabla f_k\| > \epsilon$ ;
Compute search direction
        $p_k = -H_k \nabla f_k$ ;                                    (4.16)
Set $x_{k+1} = x_k + \alpha_k p_k$ where $\alpha_k$ is computed from a line search
procedure to satisfy the Wolfe conditions ;
Define $s_k = x_{k+1} - x_k$ and $y_k = \nabla f_{k+1} - \nabla f_k$ ;
    Compute $H_{k+1}$ by means of                                (4.15);
$k \leftarrow k + 1$;
end (while)
```

The algorithm is robust, and its rate of convergence is superlinear,which is fast enough for most practical purposes.

Implementation:

A few details and enhancements need to be added to Algorithm 3.4 to produce an efficient implementation. The line search, which should satisfy either the Wolfe conditions, should always try the step length $\alpha_k = 1$ first, because this step length will eventually always be accepted (under certain conditions), there by producing superlinear convergence of the overall algorithm. The values $c_1 = 10^{-4}$ and $c_2 = 0.9$ are commonly used.

As mentioned earlier, the initial matrix H_0 often is set to some multiple βI of the identity, but there is no good general strategy for choosing the multiple β. We change the provisional value $H_0 = I$ by setting

$$H_0 \leftarrow \frac{y_k^T s_k}{y_k^T y_k} I \qquad \textbf{(4.17)}$$

before applying the update **(4.12)** , **(4.15)** to obtain H_0.

4.3 The SR1 Method:

In the BFGS and DFP updating formulae, the updated matrix B_{k+1} (or H_{k+1}) differs from its predecessor B_k (or H_k) by a rank-2 matrix. In fact, as we now show, there is a simpler rank-1 update that maintains symmetry of the matrix and allows it to satisfy the secant equation.

Unlike the rank-two update formulae, this symmetric-rank-1, or SR1, update does not guarantee that the updated matrix maintains positive definiteness. Good numerical results have been obtained with algorithms based on SR1, so we derive it here and investigate its properties.

The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma v v^T$$

Where σ is either $+1\ or -1$, and σ and v are chosen so that B_{k+1}satisfies the secant equation **(4.5)** that is, $B_{k+1}s_k = y_k$. By substituting into this equation, we obtain

$$y_k = B_k s_k + [\sigma v^T s_k]v. \qquad \textbf{(4.18)}$$

Since the term in brackets is a scalar, we deduce that v must be a multiple of $y_k - B_k s_k$, that is, $v = \delta(y_k - B_k s_k)$ for some scalar δ. By substituting this form of v into **(4.18)**, we obtain

$$(y_k - B_k s_k) = \sigma\delta^2[s_k^T\,(y_k - B_k s_k)](y_k - B_k s_k) \qquad \textbf{(4.19)}$$

and it is clear that this equation is satisfied if (and only if) we choose the parameters δ and σ to be

$$\sigma = sign\big[s_k^T\,(y_k - B_k s_k)\big], \delta = \pm\big|s_k^T\,(y_k - B_k s_k)\big|^{-\frac{1}{2}}$$

Hence, we have shown that the only symmetric rank-1 updating formula that satisfies the secant equation is given by

$$\text{(SR1)} \qquad B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \qquad \textbf{(4.20)}$$

By applying the Sherman–Morrison formula, we obtain the corresponding update formula for the inverse Hessian approximation H_k :

$$\text{(SR1)} \qquad H_{k+1} = H_k + \frac{(s_k - H_k y_k)(s_k - H_k y_k)^T}{(s_k - H_k y_k)^T y_k} \qquad \textbf{(4.21)}$$

This derivation is so simple that the SR1 formula has been rediscovered a number of times. It is easy to see that even if B_k is positive definite, B_{k+1} may not have the same property. (The same is, of course, true of H_k .) This observation was considered a major drawback in the early days of nonlinear optimization when only line search iterations were used. However, with the advent of trust-region methods, the SR1 updating formula has proved to be quite useful, and its ability to generate indefinite Hessian approximations can actually be regarded as one of its chief advantages. The main drawback of SR1 updating is that the denominator in

(4.20) or **(4.21)** can vanish. In fact, even when the objective function is a convex quadratic, there may be steps on which there is no symmetric rank-1 update that satisfies the secant equation. It pays to reexamine the derivation above in the light of this observation. By reasoning in terms of B_k (similar arguments can be applied to H_k), we see that there are three cases:

1. If $(y_k - B_k s_k)^T s_k \neq 0$,then the arguments above show that there is a unique rank-one updating formula satisfying the secant equation **(4.6)**, and that it is given by **(4.20).**
2. If $y_k = B_k s_k$, then the only updating formula satisfying the secant equation is simply $B_{k+1} = B_k$
3. If $y_k \neq B_k s_k$ and $(y_k - B_k s_k)^T s_k = 0$,then **(4.19)** shows that there is no symmetric rank-one updating formula satisfying the secant equation.

The last case clouds an otherwise simple and elegant derivation, and suggests that numerical instabilities and even breakdown of the method can occur. It suggests that rank-one updating does not provide enough freedom to develop a matrix with all the desired characteristics, and that a rank-two correction is required. This reasoning leads us back to the BFGS method, in which positive definiteness (and thus nonsingularity) of all Hessian approximations is guaranteed. We are interested in the SR1 formula for the following reasons.

(i) A simple safeguard seems to adequately prevent the breakdown of the method and the occurrence of numerical instabilities.

(ii) The matrices generated by the SR1 formula tend to be good approximations to the true Hessian matrix—often better than the BFGS approximations.

We now introduce a strategy to prevent the SR1 method from breaking down. It has been observed in practice that SR1 performs well simply by skipping the update if the denominator is small. More specifically, the update **(4.20)** is applied only if

$$\left|s_k^T(y_k - B_k s_k)\right| \geq r\|s_k\|\|y_k - B_k s_k\|, \qquad \textbf{(4.22)}$$

Where $r \in (0,1)$ is a small number, say $r = 10^{-8}$. If **(4.22)** does not hold, we set $B_{k+1} = B_k$. Most implementations of the SR1 method use a skipping rule of this kind. We now give a formal description of an SR1 method using a trust-region framework, which we prefer over a line search framework because it can accommodate indefinite Hessian approximations more easily[1,2].

Algorithm 4.2 (SR1 Trust-Region Method).

Given starting point x_0, initial Hessian approximation B_0, trust-region radius Δ_0, convergence tolerance $\epsilon > 0$, parameters $\eta \in (0, 10^{-3})$ and $r \in (0,1)$;

$k \leftarrow 0$;

```
while$\|\nabla f_k\| > \epsilon$ ;
Compute $p_k$ by solving the trust region sub-problem find $p_k$;
Compute
$y_k = \nabla f\ (x_k\ +\ p_k) - \nabla f_k$ ,
                    $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
if $\rho_k > \eta$
$x_{k+1} = x_k + p_k$;
else
$x_{k+1} = x_k$ ;
end (if)
if $\rho_k > 0.75$
if $\|p_k\| \le 0.8\Delta_k$
$\Delta_{k+1} = \Delta_k$;
else
$\Delta_{k+1} = 2\Delta_k$;
end (if)
else if $0.1 \le \rho_k \le 0.75$
$\Delta_{k+1} = \Delta_k$ ;
else
$\Delta_{k+1} = 0.5\Delta_k$;
end (if)
if $|s_k^T(y_k - B_k s_k)| \ge r\|s_k\|\|y_k - B_k s_k\|$
Use (4.20) to compute $B_{k+1}$ (even if $x_{k+1} = x_k$ );
else
$B_{k+1} \leftarrow B_k$ ;
end (if)
$k \leftarrow k + 1$;
end (while)
```

Properties of SR1 Updating:

One of the main advantages of SR1 updating is its ability to generate good Hessian approximations. We demonstrate this property by first examining a quadratic function. For functions of this type, the choice of step length does not affect the update, so to examine the effect of the updates, we can assume for simplicity a uniform step length of 1, that is,

$$p_k = -H_k \nabla f_k\ , \quad x_{k+1} = x_k + p_k\ . \qquad \textbf{(4.23)}$$

It follows that $p_k = s_k$.

Theorem 4.1.

Suppose that $f: IR^n \to IR$ is the strongly convex quadratic function $f(x) = b^T x + \frac{1}{2} x^T A x$, where A is symmetric positive definite. Then for any starting point x_0 and any symmetric starting matrix H_0, the iterates $\{x_k\}$

generated by the SR1 method **(4.21)**, **(4.23)** converge to the minimizer in at most $\boldsymbol{n}$ steps, provided that $(s_k - H_k y_k)^T y_k \neq 0$ for all k.
Moreover, if n steps are performed, and if the search directions p_i are linearly independent, then $H_n = A^{-1}$.
Theorem 4.2.
Suppose that f is twice continuously differentiable, and that its Hessian is bounded and Lipschitz continuous in a neighborhood of a point x^*. Let $\{x_k\}$ be any sequence of iterates such that $x_k \rightarrow x^*$ for some $x^* \in IR^n$. Suppose in addition that the inequality **(4.22)** holds for all k, for some $r \in (0,1)$, and that the steps s_k are uniformly linearly independent. Then the matrices B_k generated by the SR1 updating formula satisfy
$$\lim_{k\to\infty} \|B_k - \nabla^2 f(x^*)\| = 0$$
The term "uniformly linearly independent steps" means, roughly speaking, that the steps do not tend to fall in a subspace of dimension less than n. This assumption is usually, but not always, satisfied in practice[1,12].

Chapter 5

Available Methods for Derivative Free Optimization

5.1 Introduction :

Derivative free optimization methods have a long history, see ([37],[36]) and [32] for extensive discussions and references. These methods come essentially in different classes, a classification strongly influenced see [75],[15].

The first class contains algorithms which use finite-difference approximation of the objective function's derivatives in the context of a gradient based method, such as nonlinear conjugate gradients or quasi-Newton methods (see, [12],[75]). The methods in the second class are often referred to as pattern search methods, because they are based on the exploration of the variable space using a well-specified geometric pattern, typically a simplex. The algorithm described see [112] is still the most popular minimization technique in use today in this context. The methods of the third class are, for example, based on the progressive building and updating of a model of the objective function. Design of Experiment and interpolation models are proposed see ([70],[15]). Response surface methodology is described see [31]. Trust-region methods also belong to this class, see ([12],[15],[75]).

5.2 Finite-Difference Derivative Estimates:

Finite differencing refers to the estimation of $f'(x)$ using values of $f(x)$. The simplest formulas just use the difference of two function values which gives the technique its name. Finite differencing can also be applied to the calculation of $\nabla f(x)$ for multidimensional problems, as well as to the computation of $f''(x)$ and the Hessian matrix $\nabla^2 f(x)$. For a problem with n variables, computing $\nabla f(x)$ will be about n times as expensive as computing $f(x)$, and computing $\nabla^2 f(x)$ will be about n^2 times as expensive as $f(x)$. Hence, even though this technique relieves the burden of deriving and programming derivative formulas, it is expensive computationally. In addition, finite differencing only produces derivative estimates, not exact values. Finite-difference estimates can be derived from the Taylor series. In one dimension,

$$f(x+h) = f(x) + hf^{'}(x) + \frac{1}{2}h^2 f^{''}(\xi)$$

A simple rearrangement gives

$$f^{'}(x) = \frac{f(x+h)-f(x)}{h} - \frac{1}{2}hf^{''}(\xi)$$

leading to the approximation

$$f^{'}(x) = \frac{f(x+h)-f(x)}{h} \qquad \textbf{(5.1)}$$

This is the most commonly used finite-difference formula. It is sometimes called the forward difference formula because $x + h$ is a shift "forward" from the point x. This formula could also have been derived from the definition of the derivative as a limit,

$f^{'}(x) = lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$;

but this would not have provided an estimate of the error in the formula.

Example 3.3: Consider the function

$$f\ (x) = sin(x)$$

with derivative $f^{'}(x) = cos(x)$. The results of using the finite-difference formula $f^{'}(x) \approx \frac{sin\,(x\,+\,h) - sin\,(x)}{h}$

for x = 2 and for various values of h are given in Table 5.1.

The derivation of the finite-difference formula indicates that the error will be equal to $\frac{1}{2}hf^{''}(\xi)$. Since ξ is between x and $x + h$,

error$\approx \left|\frac{1}{2}hf^{''}(\xi)\right| = \left|\frac{1}{2}h(-sin(x))\right| = \left|\frac{1}{2}h(-sin(2))\right| \approx \left|\frac{1}{2}h(-0.91)\right| = 0.455h$.

This corresponds to the results in the table for h between $10^0$and 10^{-8}, but after that the error starts to increase, until eventually the finite-difference calculation estimates that the derivative is equal to zero. This phenomenon will be explained below by examining the errors that result when finite differencing is used. We now estimate the error in finite differencing when the calculations are performed on a computer. Part of the error is due to the inaccuracies in the formula itself; this is called the truncation error:

truncation error $= \frac{1}{2}h|f^{''}(\xi)|$ In addition there are rounding errors from the evaluation of the formula $(f(x+h) - f\ (x))/h$ on a computer that depend on ϵ_{mach} , the precision of the computer calculations . There are rounding errors from the evaluations of the function f in the numerator:

$$(\text{rounding error})_1 \approx\ |f\ (x)|\epsilon_{mac\,h}$$

h	$f'(x)$	Estimate	Error
10^{0}	-0.4161468365	-0.7681774187	4×10^{-1}
10^{-1}	-0.4161468365	-0.4608806017	4×10^{-2}
10^{-2}	-0.4161468365	-0.4206863500	4×10^{-3}
10^{-3}	-0.4161468365	-0.4166014158	4×10^{-4}
10^{-4}	-0.4161468365	-0.4161923007	4×10^{-5}
10^{-5}	-0.4161468365	-0.4161513830	4×10^{-6}
10^{-6}	-0.4161468365	-0.4161472912	4×10^{-7}
10^{-7}	-0.4161468365	-0.4161468813	4×10^{-8}
10^{-8}	-0.4161468365	-0.4161468392	3×10^{-9}
10^{-9}	-0.4161468365	-0.4161468947	6×10^{-8}
10^{-10}	-0.4161468365	-0.4161471167	3×10^{-7}
10^{-11}	-0.4161468365	-0.4161448963	2×10^{-6}
10^{-12}	-0.4161468365	-0.4162226119	8×10^{-5}
10^{-13}	-0.4161468365	-0.4163336342	2×10^{-4}
10^{-14}	-0.4161468365	-0.4218847493	6×10^{-3}
10^{-15}	-0.4161468365	-0.3330669073	8×10^{-2}
10^{-16}	-0.4161468365	0	4×10^{-1}

Table 5.1. Finite differencing.

which are then magnified and augmented by the division by h:

$$\text{(rounding error)}_1 \approx \frac{|f\ (x)|\epsilon_{mac\,h}}{h} + |f'\ (x)|\epsilon_{mac\,h}$$

(the first rounding error is magnified by 1/h and then there is an additional rounding error from the division that is proportional to the result $f'(x)$). Under typical circumstances, when h is small and $f(x)$ is not overly large, the first term will dominate, leading to the estimate

rounding error $\approx \frac{|f\ (x)|\epsilon_{mac\,h}}{h}$. The total error is the combination of the truncation error and the rounding error

error $\approx \frac{1}{2}h|f''\ (\xi)| + \frac{|f\ (x)|\epsilon_{mac\,h}}{h}$.

For fixed x and for almost fixed ξ (ξ is between x and $x + h$, and h will be small), this formula can be analyzed as a function of h alone.

To determine the "best" value of h we minimize the estimate of the error as a function of h. Differentiating with respect to h and setting the derivative to zero gives

$\frac{1}{2}h|f''\ (\xi)| - \frac{|f\ (x)|\epsilon_{mac\,h}}{h^2} = 0$,

which can be rearranged to give

$$h = \sqrt{\frac{2|f\ (x)|\epsilon_{mac\,h}}{|f''\ (\xi)|}}$$

In cases where $f(x)$ and $f^{''}(\xi)$ are neither especially large nor small, the simpler approximation

$$h = \sqrt{\epsilon_{mac\,h}}$$

can be used. If the more elaborate formula for h is substituted into the approximate formula for the error, then the result can be simplified to

error $\approx \sqrt{2\epsilon_{mac\,h}|f(x).f^{''}(\xi)|}$,

or more concisely to the result that the error is $O(\sqrt{\epsilon_{\mathrm{mach}}})$.

In the example above, $\epsilon_{\mathrm{mach}} \approx 10^{-16}$ and the simplified formula for h yields $h \approx \sqrt{\epsilon_{mac\,h}} \approx 10^{-8}$. This value of h gives the most accurate derivative estimate in the example.The more elaborate formula for h yields $h \approx 2.1 \times 10^{-8}$, almost the same value. The error with this value of h is about 1.4×10^{-8}, slightly worse than the value given by the simpler formula. This does not indicate that the derivation is invalid; rather it only emphasizes that the terms used in the derivation are estimates of the various errors. As expected, the errors in this example are approximately equal to$\sqrt{\epsilon_{\mathrm{mach}}}$.

In practical settings the value of $f^{''}(\xi)$ will be unknown (even the value of $f(x)$ will be unknown) and so the more elaborate formula for h cannot be used. Some software packages just use $h = \sqrt{\epsilon_{mac\,h}}$. or some simple modification of this formula (for example, taking into account $|x|$ *or* $|f(x)|$*).* An alternative is to perform extra calculations for one value of x, perhaps the initial guess for the optimization algorithm, to obtain an estimate for $f^{''}(\xi)$ and then use this to obtain a better value for h that will be used for subsequent finite-difference calculations.

An additional complication can arise if |x| is large. If $h < \epsilon_{mac\,h}|x|$, then the computed value of $x + h$ will be equal to x and the finite-difference estimate will be zero. Thus, in the general case the choice of h will depend on $\epsilon_{mac\,h}$, $|x|$, and the values of $|f^{''}|$.

If higher accuracy in the derivative estimates is required, then there are two things that can be done. One choice is to use higher-precision arithmetic (arithmetic with a smaller value of ϵ_{mach}). This might just mean switching from single to double precision, a change that can sometimes be made with an instruction to the compiler without any changes to the program. If the program is already in double precision, then on some computers it is possible to use quadruple precision, but quadruple precision arithmetic can be much slower than double precision since the instructions for it are not normally built into the computer hardware.

The other choice is to use a more accurate finite-difference formula. The simplest of these is the central-difference formula

$$f^{'}(x) = \frac{f(x+h)-f(x-h)}{2h} - \frac{1}{2}h^2[f^{'''}(\xi_1) + f^{'''}(\xi_2)].$$

It can be derived using the Taylor series for $f(x+h)$ and $f(x-h)$ about the point h. Higher derivatives can also be obtained by finite differencing. For example, the formula

$$f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2} - \frac{1}{24}h^2[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)]. \qquad \textbf{(5.2)}$$

can be derived from the Taylor series for $f(x+h)$ and $f(x-h)$ about the point x. The derivatives of multidimensional functions can be estimated by applying the finite difference formulas to each component of the gradient or Hessian matrix. If we define the vector

$$e_j = (0 \cdots 0 \;\; 1 \;\; 0 \cdots 0)^T$$

having a one in the jth component and zeroes elsewhere, then

$$[\nabla f(x)]_j \approx \frac{f(x+he_j) - f(x)}{h} \qquad \textbf{(5.3)}$$

If the gradient is known, then the Hessian can be approximated via

$$[\nabla^2 f(x)]_{jk} = \frac{\partial^2 f(x)}{\partial x_j \partial x_k} \approx \frac{[f(x+he_k) - f(x)]_j}{h}. \qquad \textbf{(5.4)}$$

If it is feasible to use complex arithmetic to evaluate $f(x)$, then an alternative way to estimate $f'(x)$ is to use

$f'(x) \approx \Im[f(x+ih)]/h,$

where $i = \sqrt{-1}$and $\Im[f]$ is the imaginary part of the function f. This formula is capable of producing more accurate estimates of the derivative (sometimes up to full machine accuracy) with only one additional function evaluation, for a broad range of values of h [1,12].

5.3 Derivative-Free Trust-Region Method for Solving Large-Scale Optimization Problems Using Truncated Newton Method and Iterative Method

In this chapter we present, a derivative-free trust-region algorithm for large-scale unconstrained optimization which are arise in many aspects of science ,engineering, and economics. In proposed method we using symmetric-rank1(SR1) discussed in chapter 4 to approximate the Hessian ,and using central finite-difference approximation to the gradient of the function .We are solving the trust-region sub-problem by two methods truncated Newton method discussed in section (3.4)and Iterative Method discussed in section (3.3).Its performance is tested on some problems.

Consider the general unconstrained optimization problem

$$min_{x \in R^n} f(x) \qquad \textbf{(5.5)}$$

Where $f(x)$ is a continuously differentiable function defined in R^n.

Many applications give rise to unconstrained optimization problems with thousands or millions of variables. Problems of this size can be solved efficiently only if the storage and computational costs of the optimization algorithm can be kept at a tolerable level. A diverse collection of large-scale optimization methods has been developed to achieve this goal, each being particularly effective for certain problem types [1],[10],[12].

Trust region methods for the unconstrained optimization problem **(5.5)** compute a trial step in each iteration. Trust-region methods make explicit reference to a “model” of the objective function. For Newton’s method this model is a quadratic model derived from the Taylor series for f about the point x_k:

$$q_k(x_k) = f(x_k) + \nabla f(x_k)^T p + \tfrac{1}{2} p^T \nabla^2 f(x_k) p \qquad \textbf{(5.6)}$$

the method will only “trust” this model within a limited neighborhood of the point x_k, defined by the constraint

$$\|p\| \leq \Delta_k \qquad \textbf{(5.7)}$$

this will serve to limit the size of the step taken from x_k to x_{k+1}. The value of k is adjusted based on the agreement between the model $q_k(p)$ and the objective function $f(x_k + p)$.If the agreement is good, then the model can be trusted and k increased. If not, then k will be decreased.(In the discussion here we assume that $\|.\| = \|.\|_2$ that is, we use the Euclidean norm. At iteration k of a trust-region method, the following sub-problem is solved to determine the step:

$$minimize_p\, q_k(x_k) = f(x_k) + \nabla f(x_k)^T p + \tfrac{1}{2} p^T \nabla^2 f(x_k) \qquad \textbf{(5.8)}$$

Subject to

$$\|p\| \leq \Delta_k$$

The following is a description of a model trust region algorithm for unconstrained optimization.

Algorithm 5.1 (A Model Trust Region Algorithm)

```
Given $\hat{\Delta} > 0$ , $\Delta_0 \in (0, \hat{\Delta})$ $and\ \eta \in \left[0, \frac{1}{4}\right]$:
For $k = 0,1,2,\ldots..$
Obtain $p_k$ by (approximately) solving (5.6);
Obtain $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
if $\rho_k < \frac{1}{4}$
        $\Delta_{k+1} = \frac{1}{4}\Delta_k$
else
if $\rho_k > \frac{3}{4}$ $and$ $\|p_k\| = \Delta_k$
        $\Delta_{k+1} = min(2\Delta_k, \hat{\Delta})$
else
        $\Delta_{k+1} = \Delta_k$
If $\rho_k < \eta$
        $x_{k+1} = x_k + p_k$
else
        $x_{k+1} = x_k$
end(for)
```

In this chapter , we study the case when the matrix $B_k = \nabla^2 f$ is updated by the symmetric-rank-1(SR1) discussed in section(3.8). When the number of variables is very large, it could be very costly to solve Equation **(5.8)** exactly. Therefore, various methods for calculating an approximate solution of Equation **(5.8)** have been developed, such as the dogleg [1] and double dogleg techniques [1], the truncated Newton Method [1,14] and subspace-iterated methods[1], etc. in this chapter we use truncated Newton Method and Iterative Method .

Truncated Newton Method:

We study truncated Newton method (modified Newton CG), which is assumed to be the solution of the trust region sub-problem **(5.6).** This algorithm, due to Steihaug [1], is specified below as Algorithm 5.2. A complete algorithm for minimizing f is obtained by using Algorithm 5.2 to generate the step p_k required by Algorithm 5.1, for some choice of tolerance ϵ_k at each iteration. we use d_j to denote the search directions of this modified CG iteration and z_j to denote the sequence of iterates that it generates[1,12,13,52,109].

<u>Algorithm 5.2 (CG-Steihaug)</u>

```
Given tolerance $\epsilon_k > 0$ ;
Set $z_0 = 0, r_0 = \nabla f_k, d_0 = -r_0 = -\nabla f_k$;
If $\|r_0\| < \epsilon_k$
Return $p_k = z_0 = 0$;
For $j = 0,1,2,\ldots..$
If $d_j^T B_k d_j \leq 0$
Find $\tau$ such that $p_k = z_j + \tau d_j$ minimizes $m_k(p_k)$ in **Algorithm 5.1** and satisfies $\|p_k\| \leq \Delta_k$
Return $p_k$;
Set $\alpha_j = r_j^T r_j / d_j^T B_k d_j$ ;
Set $z_{j+1} = z_j + \alpha_j d_j$;
If $\|z_{j+1}\| \geq \Delta_k$
Find $\tau \geq 0$ such that $p_k = z_j + \tau d_j$ satisfies $\|p_k\| = \Delta_k$;
Return $p_k$ ;
Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
If $\|r_{j+1}\| < \epsilon_k$
Return $p_k = z_{j+1}$;
Set $\beta_{j+1} = r_{j+1}^T r_{j+1} / r_j^T r_j$;
Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
End(for).
```

Iterative Solution of the Sub-Problem :
We study Iterative Solution, which is assumed to be the solution of the trust region sub-problem. A complete algorithm for minimizing f is obtained by using Algorithm 5.3 to generate the step $p_k$required by Algorithm 5.1, for some choice of tolerance ϵ_{k} at each iteration. More details in Chapter3.
Algorithm (5.3) (Trust Region Sub-problem)

```
$Given\ \lambda^{(0)}, \Delta > 0$:
    $for\ \ell = 0,1,2,\dots$
        $Factor\ B + \lambda^{(\ell)} I = R^T R;$
        $Solve\ R^T R p_\ell = -g, R^T q_\ell = p_\ell;$
        $Set$
        $\lambda^{(\ell+1)} = \lambda^{(\ell)} + \left(\frac{\|p_\ell\|}{\|q_\ell\|}\right)^2 \left(\frac{\|p_\ell\| - \Delta}{\Delta}\right);$    (5.9)
    $end\ (for).$
```

Gradient Estimation via Central Finite Differences
Finite differencing refers to the estimation of $f'(x)$ using values of $f(x)$. central finite differences **(CFD)** based on the sample set
$X = \{x + \sigma e_i\}_{i=1}^{n} \cup \{x - \sigma e_i\}_{i=1}^{n}$, and is computed as

$$[g(x)]_i = \nabla f(x) = \frac{f(x+\sigma e_i) - f(x-\sigma e_i)}{2\sigma}, \quad \text{for } i = 1, 2, \dots \dots, n \qquad \textbf{(5.10)}$$

CFD approximations require $2n$ functions evaluations. More details in section 5.2 [1][12].
The Symmetric-Rank-1(SR1) Method
We have shown that the only symmetric rank-1 updating formula that satisfies the secant equation is given by **(4.20)** the SR1 updating formula has proved to be quite useful, and its ability to generate indefinite Hessian approximations can actually be regarded as one of its chief advantages [1,12], more details in Chapter3.

The algorithm 1

The algorithm proposed using truncated Newton method to solve trust region sub-problem and using symmetric-rank-1 to find approximation to Hessian and using Central Finite Difference to approximate the Gradient.

Algorithm 5.4 (SR1 Trust-Region Method)

```
Given starting point $x_0$, initial Hessian approximation $B_0$, trust-region radius $\Delta_0$, convergence tolerance $\epsilon > 0$, parameters $\eta \in (0, 10^{-3})$ and $r \in (0,1)$;
$k \leftarrow 0$;
Compute $g_k = \nabla f_k$ using finite differences eqs(5.10)
While $\|\nabla f_k\| > \epsilon$ ;
    Compute $p_k$ by solving the subproblem using Algorithm **5.2** find $p_k$;
    Compute
    $y_k = \nabla f\ (x_k + p_k) - \nabla f_k$ ,
    $\rho_k = \dfrac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
    if $\rho_k > \eta$
        $x_{k+1} = x_k + p_k$;
    else
        $x_{k+1} = x_k$ ;
    end (if)
    if $\rho_k > 0.75$
        if $\|p_k\| \leq 0.8\Delta_k$
            $\Delta_{k+1} = \Delta_k$;
        else
            $\Delta_{k+1} = 2\Delta_k$;
        end (if)
    else if $0.1 \leq \rho_k \leq 0.75$
        $\Delta_{k+1} = \Delta_k$ ;
    else
        $\Delta_{k+1} = 0.5\Delta_k$;
    end (if)
    if $\left|s_k^T(y_k - B_k s_k)\right| \geq r\|s_k\|\|y_k - B_k s_k\|$
        Use **(4.20)** to compute $B_{k+1}$ (even if $x_{k+1} = x_k$ );
    else
        $B_{k+1} \leftarrow B_k$ ;
    end (if)
    $k \leftarrow k + 1$;
end (while)
```

The algorithm 2

The algorithm proposed using Iterative Solution method to solve trust region sub problem and using symmetric-rank-1 to find approximation to Hessian and using Central Finite Difference to approximate the Gradient.

Algorithm 5.5 (SR1 Trust-Region Method)

```
Given starting point $x_0$, initial Hessian approximation $B_0$, trust-region radius $\Delta_0$,
convergence tolerance $\epsilon > 0$, parameters $\eta \in (0, 10^{-3})$ and $r \in (0, 1)$;
$k \leftarrow 0$;
Compute $g_k = \nabla f_k$ using finite differences eqs(5.10)
While $\|\nabla f_k\| > \epsilon$ ;
Compute $p_k$ by solving the sub-problem using Algorithm5.3 find $p_k$;
Compute
$y_k = \nabla f(x_k + p_k) - \nabla f_k$ ,
        $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
if $\rho_k > \eta$
$x_{k+1} = x_k + p_k$;
else
$x_{k+1} = x_k$ ;
end (if)
if $\rho_k > 0.75$
if $\|p_k\| \leq 0.8\Delta_k$
$\Delta_{k+1} = \Delta_k$;
else
$\Delta_{k+1} = 2\Delta_k$;
end (if)
else if $0.1 \leq \rho_k \leq 0.75$
$\Delta_{k+1} = \Delta_k$ ;
else
$\Delta_{k+1} = 0.5\Delta_k$;
end (if)
if$|s_k^T(y_k - B_k s_k)| \geq r\|s_k\|\|y_k - B_k s_k\|$
Use (4.20) to compute $B_{k+1}$ (even if $x_{k+1} = x_k$ );
else
$B_{k+1} \leftarrow B_k$ ;
end (if)
$k \leftarrow k + 1$;
end (while)
```

5.4 Numerical results:

1. $f(x) = 100(x_1 - x_2)^6 + 10(x_3 - 1)^8 + (x_1 - 4)^4$ the optimal solution $x^* = [4 \quad 4 \quad 1]$

- With $x_0 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, delta = 1, iteration = 1000, erorr(\epsilon) = 0.0000001$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-6}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-6}$
iteration k	1000	60
The optimal point x^*	3.4887 4.4723 0.9812	3.9986 3.9878 1.0191

- Using $x_0 = [0 \quad 0 \quad 0]$, $delta = 1, iteration = 50000, erorr(\epsilon) = 0.0000000000001$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-12}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-12}$
iteration k	50000	1556
The optimal point x^*	4.0081 4.1612 1.0038	4.0000 4.0003 0.9998

- $\boldsymbol{x_0} = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \boldsymbol{delta} = \mathbf{1}\ , \boldsymbol{iteration} = \mathbf{10000}, \boldsymbol{erorr(\epsilon)} = \mathbf{0.0000001}$

Method	**Derivative free Trust-region method using truncated Newton method** $\epsilon = 10^{-6}$	**Derivative free Trust-region method using Iterative Solution** $\epsilon = 10^{-6}$
iteration k	**2594**	**60**
The optimal point x^*	$\mathbf{4.0632}$ $\mathbf{4.0208}$ $\mathbf{0.9812}$	$\mathbf{3.9986}$ $\mathbf{3.9878}$ $\mathbf{1.0191}$

- $\boldsymbol{x_0} = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \boldsymbol{delta} = \mathbf{1}\ , \boldsymbol{iteration} = \mathbf{10000}, \boldsymbol{erorr(\epsilon)} = \mathbf{0.00000000001}$

Method	**Derivative free Trust-region method using truncated Newton method** $\epsilon = 10^{-12}$	**Derivative free Trust-region method using Iterative Solution** $\epsilon = 10^{-12}$
iteration k	**100000**	**177**
The o ptimal point x^*	$\mathbf{4.0945}$ $\mathbf{3.8750}$ $\mathbf{1.0019}$	4.0000 4.0000 1.0140

2. $\sum_{i=1}^{10}(i-x_i)^4$ **the optimal solution** $x^* = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10]$

- Using $x_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-12}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-12}$
iteration k	10000	127
The optimal point x^*	1.0094	1.0000
	2.0015	2.0000
	3.0006	3.0000
	3.9886	4.0000
	5.0005	5.0000
	5.9993	6.0000
	7.0984	7.0000
	7.9871	8.0000
	8.6187	9.0000
	10.0007	10.0000

- Using $x_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-9}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-9}$
iteration k	10000	101
The optimal point x^*	**1.0094** **2.0015** **3.0006** **3.9886** **5.0005** **5.9993** **7.0984** **7.9871** **8.6187** **10.0007**	**1.0002** **2.0002** **3.0002** **4.0002** **5.0002** **6.0002** **7.0002** **8.0002** **8.9999** **9.9998**

3. $f(x) = \sum_{i=1}^{20}(i - x_i)^4$

With

$x_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 00 \quad 0 \quad 0]$, delta $=$ 1 , iteration $=$ 1000, erorr(ϵ) =0.00000000000001

The optimal solution

$x^* =$
$[1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \quad 1718 \quad 19 \quad 20]$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-14}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-14}$
iteration k	1000	139
The optimal point x^*	1.0000 2.0067 2.9997 3.9996 5.0067 6.0000 6.9996 8.0000 9.0003 10.0067 11.0067 11.9999 12.9997 14.0000 15.0067 16.1593 17.0004 18.0000 19.2186 20.0000	1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000 11.0000 12.0000 13.0000 14.0000 15.0000 16.0000 17.0000 18.0000 19.0000 20.0000

Conclusion

In this study a practical algorithm has been developed based on Trust Region frame work. Centered Finite Difference are used to approximate gradient of the Function, and Symmetric Rank-1 is used to approximation Hessian matrix in Trust Region frame work. The Truncated Newton method and Iterative method were used to solve the trust-region sub-problem. The results obtained from the practical implementation showed the adequacy of these methods .The method use iterative solution to solve the trust region sub-problem is more efficiency than the truncated Newton method. The work did not include constrained problems. It was limited to unconstrained Optimization problems. For further study, we suggest developing methods for constrained problems.

Appendix: MATLAB Implementation :

(i) The function lowsys()is a auxiliary function to solve the lower triangular system $Ax = b$

```
function x=lowsys(A,b)
n=length(b);
fori=1:n
x(i)=b(i);
for j=1:i-1
x(i)=x(i)-A(i,j)*x(j);
end
x(i)=x(i)/A(i,i);
end
x=x';
```

(ii) The function uppsys() is a auxiliary function to solve the upper triangular system $Ax = b$

```
function x=uppsys(A,b)
n=length(b);
fori=n:-1:1
x(i)=b(i);
for j=i+1:n
x(i)=x(i)-A(i,j)*x(j);
end
x(i)=x(i)/A(i,i);
end
x=x';
```

(iii) The function chky()obtains the choleki function L of a given positive definite matrix $(A + dI)$ or returns $p = 0$ if A is not positive definite (it uses the function lowsys())

```
function [L,p]=chky(A,d)
[n,n]=size(A);
A=A+d*eye(n);
L=[];
p=1;
if A(1,1)<=0
   p=0;
else
L(1,1)=sqrt(A(1,1));
end
k=1;
while and(k<=n-1,p==1)
   z=lowsys(L(1:k,1:k),A(k+1,1:k));
L(k+1,1:k)=z';
   c=A(k+1,k+1)-L(k+1,1:k)*L(k+1,1:k)';
if c>0
L(k+1,k+1)=sqrt(c);
else
     p=0;
end
   k=k+1;
end
```

(iv) The function cgahmed()is the mean function that obtains the best direction p at minimizing the approximating quadratic function in the region of radius delta (using truncated Newton method (modified Newton CG)).

```
[p  status ] = cgahmed(n, fk2, gk, Hk, delta,zax)
eps = 1e-10;
z = zeros(n,1);
```

```
r = gk;
d = -r;
if (norm(r) <eps)
   p = z;
status = 'Stopping criteria';
return
end
fori=1:zax
if (d'*Hk*d <= 0)
status = 'Negative curvature';
tau = roots([d'*d, 2*(d'*z), z'*z - delta^2]);
     p1 = z + tau(1)*d; % first candidate
     p2 = z + tau(2)*d; % second candidate
% Check which is largest
     m1 = fk2 + gk'*p1 + (p1'*Hk*p1)/2;
     m2 = fk2 + gk'*p2 + (p2'*Hk*p2)/2;
if (m1 < m2)
        p = p1;
return
else
        p = p2;
return
end
end
  a = (r'*d) / (d'*Hk*d);
zo = z;
  z = z + a*d;
if (norm(z) >= delta)
tau = max(roots([d'*d, d'*zo, zo'*zo - delta^2]));
assert (tau >= 0) % A positive solution should exist
     p = zo + tau*d;
status = 'Trust-region boundary';
return
end
ro = r;
  r = r + a*Hk*d;
if (norm(r) <eps)
     p = z;
status = 'Stopping criteria';
return
end
  b = (r'*r) / (ro'*ro);
  d = -r + b*d;

end
end
```

(v) The function tsubprobahmed()is the mean function that obtains the best direction *p* at minimizing the approximating quadratic function in the region of radius delta (**using Iterative Solution**).

```
p=tsubprobahmed(G,g,delta)
np=2*delta;
[L,r]=chy(G,0);
if r==1
   y=lowsys(L,-g);
   p=uppsys(L',y);
np=norm(p);
end
ifnp>delta
  d=10;
  [L,r]=chy(G,d);
while r==0
     d=2*d;
     [L,r]=chy(G,d);
end
for l=1:10
      y=lowsys(L,-g);
      p=uppsys(L',y);
      q=lowsys(L,p);
      d=d+((norm(p)/norm(q))^2)*((norm(p)-delta)/delta);
      [L,r]=chy(G,d);
while r==0
          d=d+1;
          [L,r]=chy(G,d);
end
end
end
```

(vi) The function finitedifferenceCD()is the function that obtains the Gradient using centered finite difference that using in the main symmetric rank-1 SR1 method .

```
function g=finitedifferenceCD(x)
```

```
n=length(x);
H=eye(n);
al=0.001;
hcd=3^1/3*al^1/3;
fori=1:n
   d=H(i,:);
g(:,i)=(f(x+hcd*d')-f(x-hcd*d'))/(2*hcd);
end
 g=g';
end
```

(vii) The function SR1TrustRegionnewton() input initial point x_0, initial delta, and the iterates max. (It uses all function above to give minimizing of quadratic function without using derivative.

```
[x k] = SR1TrustRegion(x0,max,delta)
%find the mimmm of the function using SRI trstregion
d=delta;
r=10^-8;
new=10^-3;
x=x0;
n=length(x);
B0=eye(n);
g=finitedifferenceCD(x);
[s  status ] = cgahmed(n, fk2, g, B0, d,max);
for k=1:max
while (norm(g) <=0.00000000001)
return
end
last_g=g;
last_x=x;
   x=x+s;
   g=finitedifferenceCD(x);
    y=g-last_g;
ared=(f(last_x)-f(x));
pred=-(((last_g)'*s)+((0.5*s')*(B0*s)));
if (ared/pred)>new
       x=x+s;
else
       x=last_x;
end
if((ared/pred)>0.75)&& ((norm(s))<=0.8*d)
```

```
delta=d;
else
delta=2*d;
end
if ((ared/pred)>0.1)&& ((ared/pred)<=0.75)
delta=d;
else
delta=0.5*d;
end
    m=B0*s;
    m1=y-m;
    m2=norm(s);
    m3=norm(m1);
    m4=m1'*s;
if (abs(s'*m1)>=r*(m2*m3))
      H=(m1*m1')/m4;
      B=H+B0;
   [s  status ] = cgahmed(n, fk2, g, B, delta,max);
      x=x+s;
end
end
end
```

the above function using truncated Newton method (**modified Newton CG**)) to solve trust region subproblem .

(viii) The function SR1TrustRegioniterative() input initial point x_0, initial delta, and the iterates max. (It uses all function above to give minimizing of quadratic function without using derivative.

```
[x k] = SR1TrustRegioniterative(x0,max,delta)
d=delta;
r=10^-8;
new=10^-3;
x=x0;
n=length(x);
B0=eye(n);
g=finitedifferenceCD(x);
s=tsubprobmarim(B0,g,d);
for k=1:max
while (norm(g) <=0.00000000001)
return
```

```
end
last_g=g;
last_x=x;
   x=x+s;
   g=finitedifferenceCD(x);
    y=g-last_g;
ared=(f(last_x)-f(x));
pred=-(((last_g)'*s)+((0.5*s')*(B0*s)));
if (ared/pred)>new
       x=x+s;
else
       x=last_x;
end
if((ared/pred)>0.75)&& ((norm(s))<=0.8*d)
delta=d;
else
delta=2*d;
end
if ((ared/pred)>0.1)&& ((ared/pred)<=0.75)
delta=d;
else
delta=0.5*d;
end
     m=B0*s;
     m1=y-m;
     m2=norm(s);
     m3=norm(m1);
     m4=m1'*s;
if (abs(s'*m1)>=r*(m2*m3))
       H=(m1*m1')/m4;
       B=H+B0;
s=tsubprobmarim(B0,g,d);
       x=x+s;
end
end
end
```

the above function using **using Iterative Solution** method to solve trust region subproblem .

References:

[1] jorge nocedal stephen j. Wright , numerical optimization ,secondedition 2006

[2] a. R. Conn, k. Scheinberg, and p. L. Toint, on the convergence of derivative-free methods for unconstrained optimization, in approximation theory and optimization: tributes to m. J. D. Powell, a. Iserles and m. Buhmann, eds., Cambridge, England, 1997, Cambridge university press, pp. 83{108}.

[3] , recent progress in unconstrained nonlinear optimization without derivatives, mathematical programming, series b, 79 (1997), pp. 397{414.

[4] , a derivative free optimization algorithm in practice, tech. Rep. Tr98/11, department of math- ematics, university of namur, namur, belgium, 1998.

[5] a. R. Conn, k. Scheinberg, and l. Vicente, error estimates and poisedness in multivariate polynomial interpolation, tech. Rep., ibm t. J. Watson research center, 2006.

[6] , geometry of interpolation sets in derivative free optimization, mathematical programming, se- ries a, 111 (2007), pp. 141{172.

[7] g. Deng and m. Ferris, adaptation of the UOBYQA algorithm for noisy functions, in proceedings of the 38th conference on winter simulation, winter simulation conference, 2006, pp. 312{319.

[8] e. D. Dolan and j. J. Mor_e, benchmarking optimization software with performance pro_les, math- ematical programming, series a, 91 (2002), pp. 201{213.

[9] r. Fourer, d. M. Gay, and b. W. Kernighan, ampl: a modeling language for mathematical programming, scienti_c press, 1993. Www.ampl.com.

[10] marazzi, m. And j. Nocedal, wedge trust region methods for derivative free optimization, mathe- matical programming, series a, 91 (2002), pp. 289{305.

[11] marco boresta1 · tommaso colombo1 · alberto de santis 1 · stefano lucidi , a mixed finite differences scheme for gradient approximation , journal of optimization theory and applications (2022) 194:124https://doi.org/10.1007/s10957-021-01994-w

[12] igor griva stephen g. Nash arielasofer george mason university fairfax, virginia, linear and nonlinear optimization second edition, society for industrial and applied mathematics, 2009

[13] yuan, y. On the truncated conjugate gradient method. Math. Program. 87, 561–573 (2000). Https://doi.org/10.1007/s101070050012

[14] a. R. Conn, n. I. M. Gould, and ph. L. Toint. Trust-region methods. Mps-siam series on optimization. Society for industrial and applied mathematics, philadelphia, pa, usa, 2000.

[14] a. R. Conn, k. Scheinberg, and ph. L. Toint. A derivative free optimization algorithm in practice, 1998. Proceedings of the 7th aiaa/usaf/nasa/issmo symposium on multidisciplinary analysis and optimization, september 2-4.

[15] a. R. Conn, p.l. toint, an algorithm using quadratic interpolation for unconstrained derivative free optimization, in "nonlinear optimization and applications", g. Di pillo and f. Gianessi, eds, plenum publishing, new york, (1996), 27-47.

[16] a. R. Conn, k. Scheinberg, p.l. toint, on the convergence of derivative-free methods for unconstrained optimization, in "approximation theory and optimization: tribute to m.j.d. powell", a. Iserles, m. Buhmann, eds, cambridge university press, (1997),83-108

[17] a. R. Conn, k. Scheinberg, p.l. toint, recent progress in unconstrained nonlinear optimization without derivatives, mathematical programming, 79 (1997), 397-414

[18] a. Waechter, an interior point algorithm for large-scale nonlinear optimization with applications in process engineering , phd. Thesis, department of chemical engineering, carnegie mellon university (2002
[19] a. R. Conn, k. Scheinberg, and ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In a. Iserles and m. Buhmann, editors, approximation theory and optimization: tributes to m. J. D. Powell, pages 83–108, cambridge, england, 1997.

[20] a. R. Conn, p.l. toint, an algorithm using quadratic interpolation for unconstrained derivative free optimization, in "nonlinear optimization and applications", g. Di pillo and f. Gianessi, eds, plenum publishing, new york, (1996), 27-47.

[21] a. R. Conn, k. Scheinberg, p.l. toint, recent progress in unconstrained nonlinear optimization without derivatives, mathematical programming, 79 (1997), 397-414

[22] a.r. conn, n.i.m. gould, ph.l. Toint, lancelot: a fortran package for large-scale nonlinear optimization, springer verlag, 1992.

[23] b. Colson, ph. L. Toint, exploiting band structure in unconstrained optimization without derivatives, a.h. siddiqi, m. Kocvera, eds, trends in industrial and applied mathematics, vol. 72 applied optimization, (2002), kluwer, 131-147.

[24] b. Colson, ph. L. Toint, a derivative-free algorithm for sparse unconstrained optimization problems, optimization and engineering, 2 (2001), 349-412

[25] b. Colson, ph. L. Toint, optimizing partially separable functions without derivatives, optimization methods and software, 20 (2005), 493-508.

[26] b. Colson, ph. L. Toint, exploiting band structure in unconstrained optimization without derivatives, a.h. siddiqi, m. Kocvera, eds, trends in industrial and applied mathematics, vol. 72 applied optimization, (2002), kluwer, 131-147

[27 b. Colson, ph. L. Toint, optimizing partially separable functions without derivatives, optimization methods and software, 20 (2005), 493-508.

[28] b. Colson, ph. L. Toint, a derivative-free algorithm for sparse unconstrained optimization problems, optimization and engineering, 2 (2001), 349-412

[29] b. Colson, ph. L. Toint, optimizing partially separable functions without derivatives, optimization methods and software, 20 (2005), 493-508

[30] ch. Audet, v. Béchard, and s. Le digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. J. Of global optimization, 41:299–318, june 2008.

[31] ch. Audet and jr. J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. Siam j. On optimization, 1]7:188–217, january 2006.

[32] convergence properties of a class of minimization algorithms, in nonlinear programming 2, o. L. Mangasarian, r. R. Meyer, and s. M. Robinson, eds., academic press, new york, 1975, pp. 1–27.

[33] carl-erik froberc instiute for computer sciences university of lund,sweden – intorduction to numerical analysis- second edition.

[34] d. Goldfarb, curvilinear path steplength algorithms for minimization which use directions of negative curvature, mathematical programming, 18 (1980), pp. 31–40

[35] d.m. gay, \computing optimal local constrained step", siam j. Sci. Stat.comp. 2(1981) 186-197.

[36] d.c. sorensen, \newton's method with a model trust region modification", siam j. Numer. Anal. 20(1982) 409-426.

[37] d. Winfield, function and functional optimization by interpolation in data tables, phd thesis, harvard university, cambridge, usa, 1969.

[38] derivative-free algorithms in engineering optimization anders holmström göteborg, october 2000 pp for the degree of master of science

[39] dixon, i.c.w. (1972). Nonlinear optimization. London, the english universities press ltd.

[40] direct search algorithms for optimization calculations, acta numerica, 7 (1998), pp. 287–336.

[41] e. O. Omojokun, trust region algorithms for optimization with nonlinear equality and inequality constraints, ph. D. Thesis, university of colorado at boulder, 1989.

[42] e. H. Moore. On the reciprocal of the general algebraic matrix. In the fourteenth western meeting of the american mathematical society, number 26, pages 394–395. Bulletin of the american mathematical society, 1920.

[43] e. Polak, optimization: algorithms and consistent approximations, springer, 1997.

[44] f. Rendl and h. Wolkowicz, \a semide¯nite framework for trust region sub-problems with applications to large scale minimization", math. Prog. 77(1997) 273-299.

[45] f. V. Berghen, h. Bersini, condor a new parallel, constrained extension of powell's uobyqa algorithm: experimental results and comparison with the dfo algorithm, journal of computational and applied mathematics, 181 (2005), 157-175

[46] g. B. Dantzig,linear programming and extensions, princeton university press, princeton, nj, 1963

[47] g. H. Golub andc. F. Vanloan, matrix computations, the johns hopkins university press, baltimore, third ed., 1996.

[48] g.h. golub and u. Von matt, \quadratically constrained least squares and quadratic problems", numerische mathematik 59(1991) 561-580.

[49] g. A. Schultz,r.b.schnabel, andr. H. Byrd, a family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties, siam journal on numerical analysis, 22 (1985), pp. 47–67.

[50] himmelblau, d. M. (1972). Applied nonlinear programming. New york, mc graw-hill.

[51] j. J. Mor´ eandd. C. Sorensen,on the use of directions of negative curvature in a modified newton method, mathematical programming, 16 (1979), pp. 1–20

[52] jorge nocedal and stephen g. Nash. A numerical study of the limited memory bfgs method and the truncated-newton method for large scale optimization. Siam journal on optimization, 1(3):358{372, 1991.

[53] j.e. dennis and h.h.w. mei, \two new unconstrained optimization algorithms which use function and gradient values", jota 28(1979) 453-482.

[54] j.j. mor¶e, \recent developments in algorithms and software for trust region methods", in: a. Bachem, m. Gräotschel and b. Korte, eds., mathematical programming: the state of the art (springer-verlag, berlin, 1983) pp. 258-287.

[55] j.j. mor'e, d.c. sorenson, computing a trust region step, siam. J. Sci. Statsit. Comput. 4 (1983), 553-572

[56] j. E. Dennis andr. B. Schnabel,a view of unconstrained optimization, in optimization, vol. 1 of handbooks in operations research and management, elsevier science publishers, amsterdam, the netherlands, 1989, pp. 1–72.

[57] j. Stoer, r. Bulirsch, introduction to numerical analysis, springer-verlag, 1993.

[58] k.schittkowski, more test examples for nonlinear programming codes, springerverlag, berlin, 1987 (numbers above 199).

[59] linear and nonlinear optimization second edition igor griva stephen g. Nash ariela sofer george mason university fairfax, virginia society for industrial and applied mathematics • philadelphia

[60] m. Bazaraa,h.sherali, andc. Shetty, nonlinear programming, theory and applications., john wiley & sons, new york, second ed., 1993.1–17.

[61] m.r. celis, j.e. dennis and r.a. tapia, \a trust region algorithm for nonlinear equality constrained optimization", in: p.t. boggs, r.h. byrd and r.b. schnabel, eds., numerical optimization (siam, philadelphia,1985) pp. 71-82.

[62] m.j.d. powell, \a new algorithm for unconstrained optimization", in: j.b. rosen, o.l. mangasarian and k. Ritter, eds., nonlinear programming (academic press, new york, 1970) pp. 31-66. (1970a)

[63] m.j.d. powell, \a hybrid method for nonlinear equations", in: p. Robinowitz, ed., numerical methods for nonlinear algebraic equations (gordon and breach science, london, 1970) pp. 87-144. (1970b)

[64] m.j.d. powell, \convergence properties of a class of minimization algorithms", in: o.l. mangasarian, r.r. meyer and s.m. robinson, eds., nonlinear programming 2 (acadmic press, new york, 1975) pp. 1-27.

[65] m.j.d. powell, \non-convex minimization calculations and the conjugate gradient method", in: d.f. griffiths, ed., numerical analysis lecturenotes in mathematics 1066 (springer-verlag, berlin, 1984) pp. 122-141.

[66] m.j.d. powell and y. Yuan, \ a trust region algorithm for equality constrained optimization", math. Prog. 49(1991) 189-211.

[67] m. J. D. Powell, a direct search optimization method that models the objective and constraint functions by linear interpolation, in "advances in optimization and numerical analysis", eds. S. Gomez and j.p. hennart, kluwer academic publishers,(1994).

[68] m. J. D. Powell. Direct search algorithms for optimization calculations. Acta numerica, 7:287–336, 1998.

[69] mitchell, t. M. (1997). Machine learning. New york, mcgraw-hill.

[70] m. Marazzi, j. Noncedal, wedge trust region methods for derivative free optimization, mathematical programming ser. A, 91 (2002), 289-305

[71] m. Schäafer, b. Karasäozen, k. Yapc, äo. Ugur, derivative free optimization of stirrer conjgurations, the proceedings of the enumath 2005 spain, july 2005, a. Bermudez, a. Gomez, d. Quintela, p. Salgado, eds, springer, 2006

[72] newton's method, in studies in numerical analysis, vol. 24 of maa studies in mathematics, the mathematical association of america, 1984, pp. 29–82.

[73] nonlinear optimization: trust region algorithms", in: s.t. xiao, f. Wu, eds., proceedings of the third chinese siam conference (tsinghua university press, beijing, 1994), pp.84-102.

[74] nash, s. G. And a. Sofer (1996). Linear and nonlinear programming. Singapore, mcgrawhill.

[75] of trust-region derivative free optimization survey methods bäulent karasäozen middle east technical university department of mathematics & institute of applied mathematics 06531 ankara, turkey.

[76] owen, a. B. (1992). “orthogonal arrays for computer experiments, integration and visualisation.” Statistica sinica2: 439-452.

[77] ,practical methods of optimization, john wiley and sons, new york, second ed. 1987.

[78] p. E. Gill,w.murray,andm. H. Wright,practical optimization, academic press, 1981.

[79] ph. L. Toint, towards an efficient sparsity exploiting newton method for minimization", in: i.s. du, ed., sparse matrices and their uses (academic press, london, 1981) pp. 57-88.

[80] p. G. Ciarlet and p. A. Raviart. General lagrange and hermite interpolation in ir^n with applications to finite element methods. Archive for rational mechanics and analysis,46(3):177–199, 1972.

[81] p. Erdős. Problems and results on the theory of interpolation. Ii. Acta mathematica hungarica, 12:235–244, 1961.

[82] polyak, b. T. (1987). Introduction to optimization. New york, optimization software inc.

[83] r.k.ahuja,t.l.magnanti, andj. B. Orlin, network flows theory, algorithms, and applications, prentice-hall, englewood cliffs, n.j., 1993.

[84] r. P. Brent, algorithms for minimization without derivatives, prentice hall, englewood cliffs, nj, 1973.

[85] r. Bulirsch andj. Stoer,introduction to numerical analysis, springer-verlag, new york, 1980.

[86] r. L. Rardin,optimization in operations research, prentice hall, englewood cliffs, nj, 1998.

[87] r. B. Schnabel ande. Eskow,a new modified cholesky factorization, siam journal on scientific computing, 11 (1991), pp. 1136–1158.

[88] r. Byrd, r.b. schnabel and g.a. shultz, a trust region algorithm for nonlinearly constrained optimization", siam j. Numer. Anal. 24 (1987) 1152-1170.

[89] r. Byrd, r.b. schnabel and g.a. shultz, approximation solution of the trust region problem by minimization over two-dimensional subspaces",math. Prog. 40(1988) 247-263.

[90] r. Fletcher, a model algorithm for composite ndo problem", math. Prog. Study 17(1982) 67-76.

[91] r. Fletcher, practical methods of optimization (second edition) (john wiley and sons, chichester, 1987)

[92] recent developments in algorithms and software for trust region methods, in mathematical programming: the state of the art, springer-verlag, berlin, 1983, pp. 258–287.

[118] r.l. burden, j.d. faires, numerical analysis, eighth ed., brooks/cole, 2005.

[93] recent progress in unconstrained nonlinear optimization without derivatives, mathematical programming, series b, 79 (1997), pp. 397–414.

[94] s. H. Cheng andn. J. Higham, a modified cholesky algorithm based on a symmetric indefinite factorization, siam journal of matrix analysis and applications, 19 (1998), pp. 1097–1100.

[95] s. Gratton, ph.l. Toint, a. Tr oltzsch technical report tr/pa/10/70an active-set trust-region method for derivative-free nonlinear bound-constrainedoptimization

[96] s. J. Smith. Lebesgue constants in polynomial interpolation. Annales mathematicae et informaticae, 33:109–123, 2006.

[97] sacks, j., h. P. Wynn, et al. (1992). "screening predicting and computer experiments." Technometrics34(1): 15-25.

[98] t. Steihaug, \the conjugate gradient method and trust regions in large scale optimization", siam j. Numer. Anal. 20(1983) 626-637.

[99] torczon, v. (1991). "on the convergence of the multidirectional search algorithm." Siam journal on optimization1(1): 123-145.

[100] the NEWUOA software for unconstrained optimization without derivatives, numerical analysis report dampt 2004/na05, university of cambridge, cambridge, uk, 2004.

[101] UOBYQA: unconstrained optimization by quadratic approximation, mathematical rogramming, series b, 92 (2002), pp. 555–582.

[102] w. L. Winston,operations research, wadsworth publishing co., third ed., 1997.

[103](http://www.lri.fr/~hansen/cmaes_inmatlab.html).
Http://www.particleswarm.info/,http://www.icsi.berkeley.edu/~storn/code

[104] w.hock and k.schittkowski, test problems for nonlinear programming codes, springerverlag, berlin, 1981 (numbers below 199) and.

[105] wilkinson householder's method for the solution of the algebraic eigen problem the computer journal, 23-27(april 1960).

[106] y. Yuan, on a subproblem of trust region algorithms for constrained optimization", math. Prog. 47(1990) 53-63.

[107] y. Yuan, on the convergence of a new trust region algorithm", numerische mathematik 70(1995) 515-539.

[108] y. Yuan, \an example of non-convergence of trust region algorithms". In: y. Yuan, ed. Advances in nonlinear programming, (kluwer, 1998), pp. 205-215

[109] y. Yuan, on the truncated conjugate gradient method", report, icm99-003, icmsec, chinese academy of sciences. Beijing, china, 1999.

[110] W. C.DAVIDON,Variable metric method for minimization, Technical ReportANL‑5990 (revised), Argonne National Laboratory, Argonne, IL, 1959.

[111] , Variable metric method for minimization, SIAM Journal on Optimization, 1 (1991), pp. 1‑17.

[112] Nelder, J. A. and R. Mead (1965). "A Simplex method for function minimization." Computer Journal7: 308-313.

Derivative-Free Trust-Region Method for Solving Large-Scale optimization Problems Using Truncated Newton Method

1. Sudan University of Science and Technology College of graduate ,Department of Mathematics, Alnajeeb177@Gmail.com
2. Department of Applied Mathematics, Faculty of Mathematical Sciences, University of Khartoum, Khartoum, Sudan , mhahashim61@gmail.com

Abstract

In this work we present a derivative-free trust-region algorithm for large-scale unconstrained optimization, using symmetric-rank1(SR1)to approximate the Hessian and using central finite-difference approximation to the gradient of the function . The truncated Newton method algorithm is used for solving the trust-region sub problem. Its performance is tested on some problems.

Keywords: Unconstrained optimization, derivative-free optimization, trust-region methods, symmetric-rank-1(SR1), Finite-difference.

1. Introduction :

The general unconstrained optimization problem is stated as :

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

where $f(x)$ is a continuously differentiable function defined in R^n.

Many applications give rise to unconstrained optimization problems with thousands or millions of variables. Problems of this size can be solved efficiently only if the storage and computational costs of the optimization algorithm can be kept at a tolerable level. A diverse collection of large-scale optimization methods has been developed to achieve this goal, each being particularly effective for certain problem types [1],[10],[13].

Trust region methods for the unconstrained optimization problem (1) compute a trial step in each iteration. Trust-region methods make explicit reference to a “model” of the objective function. For Newton’s method this model is a quadratic model derived from the Taylor series for f about the point x_k:

$$q_k(x_k) = f(x_k) + \nabla f(x_k)^T p + \tfrac{1}{2} p^T \nabla^2 f(x_k) p \tag{2}$$

the method will only "trust" this model within a limited neighborhood of the point x_k, defined by the constraint

$$\|p\| \le \Delta_k \tag{3}$$

this will serve to limit the size of the step taken from x_k to x_{k+1}. The value of k is adjusted based on the agreement between the model $q_k(p)$ and the objective function $f(x_k + p)$.If the agreement is good, then the model can be trusted and k increased. If not, then k will be decreased.(In the discussion here we assume that $\|.\| = \|.\|_2$that is, we use the Euclidean norm.

At iteration k of a trust-region method, the following subproblem is solved to determine the step:

$$minimize_p \quad q_k(x_k) = f(x_k) + \nabla f(x_k)^T p + \tfrac{1}{2} p^T \nabla^2 f(x_k) p \tag{4}$$

Subject to

$$\|p\| \le \Delta_k$$

The following is adscription of a model trust region algorithm for unconstrained optimization.

Algorithm 1 (A Model Trust Region Algorithm)

```
Given $\hat{\Delta} > 0$ , $\Delta_0 \in (0, \hat{\Delta})$ $and$ $\eta \in \left[0, \frac{1}{4}\right]$:
For $k = 0,1,2, \dots..$
Obtain $p_k$ by (approximately) solving (2);
Obtained $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
if $\rho_k < \frac{1}{4}$
```

```
        $\Delta_{k+1} = \frac{1}{4}\Delta_k$
else
if $\rho_k > \frac{3}{4}$ $and$ $\|p_k\| = \Delta_k$
        $\Delta_{k+1} = min(2\Delta_k, \hat{\Delta})$
else
      $\Delta_{k+1} = \Delta_k$
If $\rho_k < \eta$
      $x_{k+1} = x_k + p_k$
else
      $x_{k+1} = x_k$
end(for)
```

In this paper, we study the case when the matrix $B_k = \nabla^2 f$ is updated by the symmetric-rank-1(SR1). The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma v v^T \qquad (5)$$

where σ is either $+1$ or -1, and σ and v are chosen so that B_{k+1} satisfies the (secant equation), that is,

$$y_k = B_{k+1} s_k \qquad (6)$$

When the number of variables is very large, it could be very costly to solve Eqs.(4) exactly. Therefore, various methods for calculating an approximate solution of Eqs. (4) have been developed, such as the dogleg and double dogleg techniques [1], the truncated CG method[1,14] and subspace-iterated methods [1], to name few.

2. Modified Newton CG Algorithm:

In this section, we study truncated Newton method (**modified Newton CG**), which is assumed to be the solution of the trust region sub problem (4). This algorithm, due to Steihaug [1], is specified below as **Algorithm 2**. A complete algorithm for minimizing f is obtained by using **Algorithm 2** to generate the step p_k required by **Algorithm 1**, for some choice of tolerance ϵ_k at each iteration. we use d_j to denote the search directions of this modified CG iteration and z_j to denote the sequence of iterates that it generates[1,14].

Algorithm 2 (CG-Steihaug)

```
Given tolerance $\epsilon_k > 0$ ;
Set $z_0 = 0, r_0 = \nabla f_k, d_0 = -r_0 = -\nabla f_k$;
If $\|r_0\| < \epsilon_k$
Return $p_k = z_0 = 0$;
For $j = 0,1,2,\dots..$
If ${d^T}_j B_k d_j \le 0$
Find $\tau$ such that $p_k = z_j + \tau d_j$ minimizes $m_k(p_k)$ in Algorithm 1 and satisfies $\|p_k\| \le \Delta_k$
Return $p_k$;
Set $\alpha_j = {r^T}_j r_j / {d^T}_j B_k d_j$ ;
Set $z_{j+1} = z_j + \alpha_j d_j$;
If $\|z_{j+1}\| \ge \Delta_k$
Find $\tau \ge 0$ such that $p_k = z_j + \tau d_j$ satisfies $\|p_k\| = \Delta_k$;
Return $p_k$ ;
Set $r_{j+1} = r_j + \alpha_j B_k d_j$;
If $\|r_{j+1}\| < \epsilon_k$
Return $p_k = z_{j+1}$;
Set $\beta_{j+1} = {r^T}_{j+1} r_{j+1} / {r^T}_j r_j$;
Set $d_{j+1} = -r_{j+1} + \beta_{j+1} d_j$;
End(for)
```

3. Gradient Estimation Via Central Finite Differences

Finite differencing refers to the estimation of $f^{'}(x)$ using values of $f(x)$. central finite differences **(CFD)** based on the sample set $X = \{x +$ *σei}i=1n∪{x−σei}i=1m*n , and is computed as

$$[g(x)]_i = \nabla f(x) = \frac{f(x+\sigma e_i)-f(x-\sigma e_i)}{2\sigma}, \text{for } i = 1, 2 , \dots \dots \dots , n \tag{7}$$

CFD approximations require $2n$ functions evaluations[1][12][13].

4. the symmetric-rank-1(SR1) method

The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma v v^T \tag{8}$$

where σ is either $+1$ or -1, and σ and v are chosen so that B_{k+1} satisfies the eqs(6) . By substituting into this equation, we obtain

$$y_k = B_k s_k + [\sigma v^T s_k] v. \tag{9}$$

Since the term in brackets is a scalar, we deduce that v must be a multiple of $y_k - B_k s_k$, that is, $v = \delta(y_k - B_k s_k)$ for some scalar δ. By substituting this form of v into (7), we obtain

$$(y_k - B_k s_k) = \sigma \delta^2 [s^T{}_k (y_k - B_k s_k)](y_k - B_k s_k), \tag{10}$$

and it is clear that this equation is satisfied if (and only if) we choose the parameters δ and σ to be

$$\sigma = sign[s^T{}_k (y_k - B_k s_k)] , \delta = \pm |s^T{}_k (y_k - B_k s_k)|^{-1/2} \tag{11}$$

Hence ,we have shown that the only symmetric rank-1 updating formula that satisfies the secant equation is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \tag{12}$$

the SR1 updating formula has proved to be quite useful, and its ability to generate indefinite Hessian approximations can actually be regarded as one of its chief advantages. give a formal description of an SR1 method using a trust-region framework[1,13,]

5. The Algorithm

Due to the truncated Newton method (modified Newton CG)properties studied in the section2, and central Finite difference (CFD) studied in section3, and the symmetric-rank-1(SR1) method studied in section3, we can construct trust region algorithms based on the traditional trust region philosophy.

The algorithm proposed using truncated Newton method to solve trust region subproblem and using symmetric-rank-1 to find approximation to Hessian and using finite difference to approximate the Gradient .

Algorithm 4 (SR1 Trust-Region Method)

```
Given starting point $x_0$, initial Hessian approximation $B_0$, trust-region radius $\Delta_0$, convergence tolerance $\epsilon > 0$, parameters $\eta \in (0, 10^{-3})$ and $r \in (0,1)$;
$k \leftarrow 0$;
Compute $g_k = \nabla f_k$ using finite differences eqs(7)
while $\|\nabla f_k\| > \epsilon$ ;
Compute $p_k$ by solving the subproblem using Algorithm 2 find $p_k$;
Compute
$y_k = \nabla f\ (x_k + p_k) - \nabla f_k$ ,
        $\rho_k = \frac{f(x_k)-f(x_k+p_k)}{m_k(0)-m_k(p_k)}$
if $\rho_k > \eta$
$x_{k+1} = x_k + p_k$ ;
else
$x_{k+1} = x_k$ ;
end (if)
if $\rho_k > 0.75$
if $\|p_k\| \le 0.8\Delta_k$
$\Delta_{k+1} = \Delta_k$;
```

```
else
$\Delta_{k+1}= 2\Delta_k$;
end (if)
else if $0.1 \leq \rho_k \leq 0.75$
$\Delta_{k+1}= \Delta_k$ ;
else
$\Delta_{k+1}= 0.5\Delta_k$;
end (if)
if  $|s^T{}_k(y_k - B_k s_k)| \geq r\|s_k\| \, \|y_k - B_k s_k\|$
Use (12) to compute $B_{k+1}$ (even if $x_{k+1} = x_k$ );
else
$B_{k+1} \leftarrow B_k$ ;
end (if)
$k \leftarrow k + 1$;
end (while)
```

6. Numerical Results

4. $\boldsymbol{f(x) = 100(x_1 - x_2)^6 + 10(x_3 - 1)^8 + (x_1 - 4)^4}$ **optimal solution** $\boldsymbol{x^* = [4 \quad 4 \quad 1]}$

- **With** $\boldsymbol{x_0 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, delta = 1\,, iteration = 1000, erorr(\epsilon) = 0.0000001}$
- **Solution with** `SR1TrustRegionnewton()`

$$\boldsymbol{x = \begin{bmatrix} 3.4887 \\ 4.4723 \\ 0.9812 \end{bmatrix} \;, k = 1000}$$

- **With** $\boldsymbol{x_0 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, delta = 1\,, iteration = 10000, erorr(\epsilon) = 0.0000001}$
- **Solution with** `SR1TrustRegionnewton()`

$$\boldsymbol{x = \begin{bmatrix} 4.0632 \\ 4.0208 \\ 0.9812 \end{bmatrix} \;, k = 2594}$$

- $\boldsymbol{x_0 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, delta = 1\,, iteration = 1000, erorr(\epsilon) = 0.0001}$
- **Solution with** `SR1TrustRegionnewton()`

$$\boldsymbol{x} = \begin{bmatrix} 4.0170 \\ 3.9608 \\ 1.0756 \end{bmatrix} , \qquad \boldsymbol{k} = \mathbf{164}$$

5. $\sum_{i=1}^{10}(i - x_i)^4$ **the optimal solution** <u>$x^* = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10]$</u>

- **With** $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{delta} = \mathbf{1}, \boldsymbol{iteration} = \mathbf{1000}, \boldsymbol{erorr}(\epsilon) = \mathbf{0.0001}$
- **Solution with** `SR1TrustRegionnewton()`

$$\boldsymbol{x} = \begin{bmatrix} 1.0017 \\ 2.0013 \\ 3.3722 \\ 3.9957 \\ 5.0033 \\ 6.0018 \\ 7.0023 \\ 8.0019 \\ 9.0008 \\ 9.9998 \end{bmatrix}, \boldsymbol{k} = \mathbf{64}$$

- **With** $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{delta} = \mathbf{1}, \boldsymbol{iteration} = \mathbf{1000}, \boldsymbol{erorr}(\epsilon) = \mathbf{0.0000001}$
- **Solution with** `SR1TrustRegionnewton()`

$$x = \begin{bmatrix} 0.9954 \\ 1.9923 \\ 3.2185 \\ 3.8748 \\ 5.2493 \\ 5.9963 \\ 7.0000 \\ 7.9981 \\ 8.9956 \\ 10.0002 \end{bmatrix}, k = 82$$

- With $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, delta = 1, iteration = 1000, erorr(\epsilon) = 0.00000001$
- **Solution with** `SR1TrustRegionnewton()`
- $x = \begin{bmatrix} 1.0293 \\ 2.0592 \\ 2.9795 \\ 3.6999 \\ 4.9932 \\ 5.9282 \\ 6.9937 \\ 8.0713 \\ 9.0301 \\ 9.9572 \end{bmatrix}, k = 465$

- With $x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, delta = 1, iteration = 10000, erorr(\epsilon) = 0.0000000001$
- **Solution with** `SR1TrustRegionnewton()`

$$x = \begin{bmatrix} 1.0094 \\ 2.0015 \\ 3.0006 \\ 3.9886 \\ 5.0005 \\ 5.9993 \\ 7.0984 \\ 7.9871 \\ 8.6187 \\ 10.0007 \end{bmatrix}, k = 10000$$

7. Conclusion

In this paper we have presented a derivative-free trust-region algorithm for large-scale unconstrained optimization. The algorithm using symmetric-rank1(SR1)to approximation Hessian and using centered finite-difference approximation to the gradient of function , and using the truncated Newton method algorithm for solving the trust-region sub problem. Numerical experiments on several functions show the good performances of the proposed method.

References

[1] Jorge Nocedal Stephen J. Wright , Numerical Optimization ,SecondEdition 2006
[2] A. R. Conn, K. Scheinberg, and P. L. Toint, On the convergence of derivative-free methods for unconstrained optimization, in Approximation Theory and Optimization: Tributes to M. J. D. Powell, A. Iserles and M. Buhmann, eds., Cambridge, England, 1997, Cambridge University Press, pp. 83{108.
[3] , Recent progress in unconstrained nonlinear optimization without derivatives, Mathematical Pro-
gramming, Series B, 79 (1997), pp. 397{414.
[4] , A derivative free optimization algorithm in practice, Tech. Rep. TR98/11, Department of Math- ematics, University of Namur, Namur, Belgium, 1998.
[5] A. R. Conn, K. Scheinberg, and L. Vicente, Error estimates and poisedness in multivariate
polynomial interpolation, tech. rep., IBM T. J. Watson Research Center, 2006.
[6] , Geometry of interpolation sets in derivative free optimization, Mathematical Programming, Se- ries A, 111 (2007), pp. 141{172.
[7] G. Deng and M. Ferris, Adaptation of the UOBYQA algorithm for noisy functions, in Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference, 2006, pp. 312{319.

[8] E. D. Dolan and J. J. Mor_e, Benchmarking optimization software with performance pro_les, Math- ematical Programming, Series A, 91 (2002), pp. 201{213.
[9] R. Fourer, D. M. Gay, and B. W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Scienti_c Press, 1993. www.ampl.com.
[10] Marazzi, M. and J. Nocedal, Wedge trust region methods for derivative free optimization, Mathe- matical Programming, Series A, 91 (2002), pp. 289{305.
[11] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, Trust-Region Methods, MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2000.
[12] Marco Boresta1 · Tommaso Colombo1 · Alberto De Santis 1 · Stefano Lucidi , A Mixed Finite Differences Scheme for Gradient Approximation , Journal of Optimization Theory and Applications (2022) 194:124https://doi.org/10.1007/s10957-021-01994-w
[13] Igor Griva Stephen G. Nash Ariela Sofer George Mason University Fairfax, Virginia, Linear and Nonlinear Optimization second edition, Society for Industrial and Applied Mathematics, 2009
[14] Yuan, Y. On the truncated conjugate gradient method. Math. Program. 87, 561–573 (2000). https://doi.org/10.1007/s101070050012
[15]S. M. Robinson, *Numerical Optimization 2nd ed (Jorge Nocedal, Stephen J.Wright).pdf*.

[16]B. Karasözen, "Survey of trust-region derivative free optimization methods," J. Ind. Manag. Optim., vol. 3, no. 2, pp. 321–334, 2007, doi: 10.3934/jimo.2007.3.321.
[17]E. A. E. Gumma, M. M. Ali, and M. H. A. Hashim, "A derivative-free algorithm for non-linear optimization with linear equality constraints," Optimization, vol. 69, no. 6, pp. 1361–1387, 2020, doi: 10.1080/02331934.2019.1690491.
[18]M. J. D. Powell, "The NEWUOA software for unconstrained optimization without derivatives," pp. 255–297, 2006, doi: 10.1007/0-387-30065-1_16.

Derivative-Free Trust-Region Method for Solving Large-Scale Optimization Problems Using Iterative Method

Ahmed Mohamed[1] and M. H. A. Hashim[2]

3. Sudan University of Science and Technology ,College of graduate ,Department of Mathematics, Alnajeeb177@Gmail.com
4. Department of Applied Mathematics, Faculty of Mathematical Sciences, University of Khartoum, Khartoum, Sudan , mhahashim61@gmail.com

Abstract

In this paper, we present a derivative-free trust-region algorithm for large-scale unconstrained optimization, using symmetric-rank1(SR1)to approximate the Hessian. The centeral finite-differences are used to approximate the gradient of the function. An Iterative Solution Method is used for solving the trust-region sub-problem. Its performance is tested on some problems and Compared to the Truncated Newton Method for solving trust-region sub-problem.

Keywords : Unconstrained Optimization, Derivative-Free Optimization, Trust-Region Methods, Symmetric-Rank-1(SR1), Finite-Differences.

1. Introduction

The general unconstrained optimization problem is stated as

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

Where f(x) is a continuously differentiable function defined in R^n.
large-scale optimization methods has been developed to solve unconstrained optimization problems with thousands or millions of variables which arise in many applications . Problems of this size can be solved efficiently only if the storage and computational costs of the optimization algorithm can be kept at a tolerable level. To achieve this goal collection of large-scale optimization methods has been developed, each being particularly effective for certain problem types[1][10][13].

Trust region methods for the unconstrained optimization problem (1) compute a trial step in each iteration by solving the following sub-problem

$$\text{minimize}_p \quad q_k(x_k) = f(x_k) + \nabla f(x_k)^T p + \frac{1}{2} p^T \nabla^2 f(x_k) p \tag{2}$$

Subject to

$$\|p\| \leq \Delta_k$$

The following is a description of a model trust region algorithm for unconstrained optimization.

Algorithm 1 (A Model Trust Region Algorithm)

```
Given $\hat{\Delta} > 0$ , $\Delta_0 \in (0, \hat{\Delta})$ and $\eta \in \left[0, \frac{1}{4}\right]$:
For $k = 0,1,2, \ldots ..$
Obtain $p_k$ by approximately  solving (2);
Obtain  $\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(0) - m_k(p_k)}$
If $\rho_k < \frac{1}{4}$
        $\Delta_{k+1} = \frac{1}{4} \Delta_k$
else
if $\rho_k > \frac{3}{4}$ and $\|p_k\| = \Delta_k$
        $\Delta_{k+1} = \min(2\Delta_k, \hat{\Delta})$
else
        $\Delta_{k+1} = \Delta_k$
If $\rho_k < \eta$
        $x_{k+1} = x_k + p_k$
else
        $x_{k+1} = x_k$
end(for)
```

In this paper, we study the case when the matrix $B_k = \nabla^2 f$ is updated by the symmetric-rank-1(SR1) method at each iteration . The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma vv^T \tag{3}$$

where σ is either $+1$ or -1, and σ and v are chosen so that B_{k+1} satisfies the (secant equation), that is,

$$y_k = B_{k+1}s_k \tag{4}$$

When the number of variables is very large, it could be very costly to solve Eqs(2) exactly. Therefore, various methods for calculating an approximate solution of Eqs(2) have been developed, such as the dogleg and double dogleg techniques ,the truncated CG method[1,14] and subspace-iterated methods [1], to name few.

2. Iterative Solution Of The Sub-problem

In this section, we study Iterative Solution, The technique that uses the characterization

$$(B + \lambda I)p^* = -g \tag{5}$$

for some $\lambda \geq 0$.

of the sub-problem solution, applying Newton's method to find the value of λ which matches the given trust-region radius Δ in (2).

We Define

$$p(\lambda) = -(B + \lambda I)^{-1}g$$

For λ sufficiently large that $B + \lambda I$ is positive definite and seek a value $\lambda > 0$ such that

$$\|p(\lambda)\| = \Delta \tag{6}$$

This problem is a one-dimensional root-finding problem in the variable λ. The method is assumed to be the solution of the trust region sub-problem (2). A complete algorithm for minimizing f is obtained by using Algorithm 1 to generate the step p_k required by Algorithm1, for some choice of tolerance ϵ_k at each iteration.

Algorithm 2 (Trust Region Sub-Problem)

```
Given $\lambda^{(0)}, \Delta > 0$:
 for $\ell = 0,1,2,\dots$
        Factor $B + \lambda^{(\ell)} I = R^T R$;
                Solve $R^T R p_\ell = -g, R^T q_\ell = p_\ell$;
            Set
            $\lambda^{(\ell+1)} = \lambda^{(\ell)} + \left(\frac{\|p_\ell\|}{\|q_\ell\|}\right)^2 \left(\frac{\|p_\ell\| - \Delta}{\Delta}\right)$;        (7)
end (for).
```

3. Gradient Estimation Via Central Finite Differences

Finite differencing refers to the estimation of $f'(x)$ using values of $f(x)$. central finite differences (CFD) based on the sample set $X = \{x + \sigma e_i\}_{i=1}{}^n \cup \{x - \sigma e_i\}_{i=1}{}^n n$, and is computed as

$$[g(x)]_i = \nabla f(x) = \frac{f(x+\sigma e_i) - f(x-\sigma e_i)}{2\sigma}, \text{ for } i = 1, 2, \dots\dots\dots, n \tag{8}$$

CFD approximations require 2n functions evaluations[1][12][13].

4. The Symmetric-Rank-1(SR1) Method

The symmetric rank-1 update has the general form

$$B_{k+1} = B_k + \sigma v v^T \tag{9}$$

where σ is either $+1$ or -1, and σ and v are chosen so that B_{k+1} satisfies the eqs(4) . By substituting into this equation, we obtain

$$y_k = B_k s_k + [\sigma v^T s_k] v \tag{10}$$

Since the term in brackets is a scalar, we deduce that v must be a multiple of $y_k - B_k s_k$, that is, $v = \delta (y_k - B_k s_k)$ for some scalar δ . By substituting this form of v into (7), we obtain

$$(y_k - B_k s_k) = \sigma\, \delta^{\,2}[s^T{}_k(y_k - B_k s_k)](y_k - B_k s_k), \qquad (11)$$

and it is clear that this equation is satisfied if (and only if) we choose the parameters δ and σ to be

$$\sigma = \mathrm{sign}[s^T{}_k(y_k - B_k s_k)]\,, \delta = \pm\left|s^T{}_k(y_k - B_k s_k)\right|^{-1/2} \qquad (12)$$

Hence ,we have shown that the only symmetric rank-1 updating formula that satisfies the secant equation is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} \qquad (13)$$

the SR1 updating formula has proved to be quite useful, and its ability to generate indefinite Hessian approximations can actually be regarded as one of its chief advantages. give a formal description of an SR1 method using a trust-region framework[1,13,]

5. The Algorithm

Due to the Iterative method properties studied in the section2, and central Finite difference (CFD) studied in section3, and the symmetric-rank-1(SR1) method studied in section4, we can construct trust region algorithms based on the traditional trust region philosophy. The algorithm proposed using Iterative method to solve trust region sub-problem and using symmetric-rank-1 to update Hessian matrix at each iteration and using finite difference to approximate the Gradient of the function .

Algorithm 3 (SR1 Trust-Region Method)

```
Given starting point $x_0$, initial Hessian approximation $B_0$, trust-region radius $\Delta_0$, convergence tolerance $\epsilon > 0$, parameters $\eta \in (0, 10^{-3})$ and $r \in (0,1)$;
$k \leftarrow 0$;
Compute $g_k = \nabla f_k$ using finite differences eqs(8)
while $\|\nabla f_k\| > \epsilon$ ;
Compute $p_k$ by solving the subproblem using Algorithm 2 find $p_k$;
```

```
Compute
$y_k = \nabla f(x_k + p_k) - \nabla f_k$ ,
        $\rho_k = \frac{f(x_k)-f(x_k+p_k)}{m_k(0)-m_k(p_k)}$
if $\rho_k > \eta$
$x_{k+1} = x_k + p_k$ ;
else
$x_{k+1} = x_k$ ;
end (if)
if $\rho_k > 0.75$
if $\|p_k\| \le 0.8\Delta_k$
$\Delta_{k+1} = \Delta_k$;
else
$\Delta_{k+1} = 2\Delta_k$;
end (if)
else if $0.1 \le \rho_k \le 0.75$
$\Delta_{k+1} = \Delta_k$ ;
else
$\Delta_{k+1} = 0.5\Delta_k$;
end (if)
if  $\left|s^T{}_k(y_k - B_k s_k)\right| \ge r\|s_k\| \, \|y_k - B_k s_k\|$
Use (13) to compute $B_{k+1}$ (even if $x_{k+1} = x_k$ );
else
$B_{k+1} \leftarrow B_k$ ;
end (if)
$k \leftarrow k + 1$;
end (while)
```

6. Numerical Results:

1- $\boldsymbol{f(x)} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{10}}(\boldsymbol{i} - \boldsymbol{x_i})^{\boldsymbol{4}}$ the optimal solution

$\boldsymbol{x^*} = [\mathbf{1} \quad \mathbf{2} \quad \mathbf{3} \quad \mathbf{4} \quad \mathbf{5} \quad \mathbf{6} \quad \mathbf{7} \quad \mathbf{8} \quad \mathbf{9} \quad \mathbf{10}]$, $\boldsymbol{erorr\ (\epsilon)} = \mathbf{10^{-12}}$

Using $x_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-12}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-12}$
iteration k	10000	127
The optimal point x^*	1.0094 2.0015 3.0006 3.9886 5.0005 5.9993 7.0984 7.9871 8.6187 10.0007	1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000

2- $\boldsymbol{f(x)} = \mathbf{100}(\boldsymbol{x_1} - \boldsymbol{x_2})^{\mathbf{6}} + \mathbf{10}(\boldsymbol{x_3} - \mathbf{1})^{\mathbf{8}} + (\boldsymbol{x_1} - \mathbf{4})^{\mathbf{4}}$ the optimal solution

$\boldsymbol{x^*} = [\mathbf{4} \quad \mathbf{4} \quad \mathbf{1}]$, $\boldsymbol{erorr\ (\epsilon)} = \mathbf{10^{-12}}$

Using $x_0 = [0 \quad 0 \quad 0]$, $iteration = 10000$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-12}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-12}$
iteration k	10000	610
The optimal point x^*	4.1286 4.3041 1.0038	4.0000 3.9994 0.9897

- Using $x_0 = [0 \quad 0 \quad 0]$, $iteration \quad = 50000$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-12}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-12}$
iteration k	50000	1556
The optimal point x^*	4.0081 4.1612 1.0038	4.0000 4.0003 0.9998

3- $f(x) = \sum_{i=1}^{20} (i - x_i)^4$

With

$x_0 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 00 \quad 0 \quad 0]$, delta $=$ 1 , iteration $= 1000$, erorr(ϵ) $=$0.00000000000001

The optimal solution

$x^* =$

$[1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \quad 13 \quad 14 \quad 15 \quad 16 \quad 17 \; 18 \quad 19 \quad 20]$

Method	Derivative free Trust-region method using truncated Newton method $\epsilon = 10^{-14}$	Derivative free Trust-region method using Iterative Solution $\epsilon = 10^{-14}$
iteration k	1000	139
The optimal point x^*	1.0000 2.0067 2.9997 3.9996 5.0067 6.0000 6.9996 8.0000 9.0003 10.0067 11.0067 11.9999 12.9997 14.0000	1.0000 2.0000 3.0000 4.0000 5.0000 6.0000 7.0000 8.0000 9.0000 10.0000 11.0000 12.0000 13.0000 14.0000

	15.0067 16.1593 17.0004 18.0000 19.2186 20.0000	15.0000 16.0000 17.0000 18.0000 19.0000 20.0000

7. Conclusions

In this paper we have presented derivative-free trust-region algorithm for large-scale unconstrained optimization. The algorithm using symmetric-rank1(SR1)to approximation Hessian and using centeral finite-difference approximation to the gradient of function. The Iterative method is used for solving the trust-region sub-problem ,and the results is compared with Truncated Newton method. Numerical experiments on several functions show the good performance of the proposed method. For further study, we suggest developing methods for constrained problems.

References:

[1] Jorge Nocedal Stephen J. Wright , Numerical Optimization ,SecondEdition 2006

[2] A. R. Conn, K. Scheinberg, and P. L. Toint, On the convergence of derivative-free methods for unconstrained optimization, in Approximation Theory and Optimization: Tributes to M. J. D. Powell, A. Iserles and M. Buhmann, eds., Cambridge, England, 1997, Cambridge University Press, pp. 83{108.

[3] , Recent progress in unconstrained nonlinear optimization without derivatives, Mathematical Pro- gramming, Series B, 79 (1997), pp. 397{414.

[4] , A derivative free optimization algorithm in practice, Tech. Rep. TR98/11, Department of Math- ematics, University of Namur, Namur, Belgium, 1998.

[5] A. R. Conn, K. Scheinberg, and L. Vicente, Error estimates and poisedness in multivariate polynomial interpolation, tech. rep., IBM T. J. Watson Research Center, 2006.

[6] , Geometry of interpolation sets in derivative free optimization, Mathematical Programming, Se- ries A, 111 (2007), pp. 141{172.

[7] G. Deng and M. Ferris, Adaptation of the UOBYQA algorithm for noisy functions, in Proceedings of the 38th conference on Winter simulation, Winter Simulation Conference, 2006, pp. 312{319.

[8] E. D. Dolan and J. J. Mor_e, Benchmarking optimization software with performance pro_les, Math- ematical Programming, Series A, 91 (2002), pp. 201{213.

[9] R. Fourer, D. M. Gay, and B. W. Kernighan, AMPL: A Modeling Language for Mathematical Programming, Scienti_c Press, 1993. www.ampl.com.

[10] Marazzi, M. and J. Nocedal, Wedge trust region methods for derivative free optimization, Mathe- matical Programming, Series A, 91 (2002), pp. 289{305.

[11] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, Trust-Region Methods, MPS-SIAM Series on Optimization, SIAM, Philadelphia, 2000.

[12] Marco Boresta1 · Tommaso Colombo1 · Alberto De Santis 1 · Stefano Lucidi , A Mixed Finite Differences Scheme for Gradient Approximation , Journal of Optimization Theory and Applications (2022) 194:124https://doi.org/10.1007/s10957-021-01994-w

[13] Igor Griva Stephen G. Nash Ariela Sofer George Mason University Fairfax, Virginia, Linear and Nonlinear Optimization second edition, Society for Industrial and Applied Mathematics, 2009

[14] Yuan, Y. On the truncated conjugate gradient method. Math. Program. 87, 561–573 (2000). https://doi.org/10.1007/s101070050012

[15]"Linear Algebra and Its Applications," Linear Algebra and Its Applications, vol. 373, no. SUPPL. 2003, doi: 10.2307/2978065.

[16]S. Hammarling and C. Lucas, "Updating the QR factorization and the least squares problem," MIMS Rep., no. November, p. 73, 2008, [Online]. Available: http://eprints.ma.man.ac.uk/1192/.