Implementation of Trust Region Methods in Optimization

By
Mohammed Omar El-Hajj
BS, Beirut University College, 1995

PROJECT

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science at
Lebanese American University
May 1998

Signatures Redacted

Dr. Issam Moghrabi (Advisor)
Assistant Professor of Computer Science
Lebanese American University

Signatures Redacted

Dr. May Abboud
Associate Professor of Mathematics and Computer Science
Lebanese American University

To Palestinian martyrs, and to my parents

TABLE OF CONTENTS

Symbols Notation and abbreviation

Superscript "T"	Transposition
x_{min}	The solution of the minimisation problem
x_i	i^{th} estimate of x_{min}
$\|.\|$	The Euclidean norm of a vector , that is $\|x\|_2 = \left[\sum_{i=1}^{n} x_i^2\right]^{1/2}$
$f(x)$	The function to be minimised
$g(x)$	The gradient of $f(x)$, that is $[g(x)]^T = [\partial f/\partial x_1 , \partial f/\partial x_2, \ldots, \partial f/\partial x_n]$
G_i or $g(x_i)$	The gradient evaluated at x_I
s_i	$x_{i+1} - x_I$
y_i	$g_{i+1} - g_i$
$G(x)$ or G	The n by n hessian matrix of f(x), whose (i,j)th element is given by $\partial^2 f(x) / \partial x_i \partial x_j$
B_i	The i^{th} approximations to the Hessian matrix
H_i	The i^{th} approximations to the inverse Hessian matrix
R	The set of real numbers
p_i	The Sreach direction on iteration i
t_i	The step length on iteration i
Q-N	An abbreviation for "Quasi – Newton "
$\{x_i\}$	The sequence $x_0, x_1, x_2, \ldots$

AKNOWLEDGMENTS

I am grateful to my advisor Dr Issam Moghrabi for his encouragment, kind advice, contribution of a mass of a very useful comments, critisism, and suggestions. Thus, without his help this project would never have been brought into light. Thanks are also due to Dr May Abboud, my reader for her valuable comments. Special thanks also go to my best friend Mazen Habbal for his assistance in to completing this work.

CURRICULUM VITAE

NAME	:	Hajj, Mohammad Omar 915044.
DATE & PLACE OF BIRTH	:	- Kuwait 22-2-1973.
ADDRESS	:	- Lebanon, Saida, Abra.
NATIONALITY	:	- Palestinian.
EDUCATION	:	- BS degree in Computer Science BUC 1995 with Honor . - MS degree in Computer Science LAU 1998. - Teaching Diploma 1998 (UNESCO).
EXPERIENCE	:	- 1994 – 1998 Software Engineering Mikan Computer Systems - 1995 – Present UNRWA , Technical Instructor Computer Science
QUALIFICATIONS	:	- MS-office Expert - Programming using Oracle, FoxPro, Access, Clipper, C^{+}. - Good Knowledge in Network installation, and maintenance.
REFERENCES	:	- Dr Issam Moghrabi, Assistant Professor of Computer Science - Dr May Abboud, Associate Professor of Mathematics and Computer Science - Dr Nasha't Mansour, Assistant Professor of Computer Science

ABSTRACT

This project presents a new approach to Quasi-Newton methods for unconstrained optimization. Quasi-Newton Methods update at each iteration the existing Hessian approximation (or its inverse) cheaply by integrating data derived from the previously completed one, which is soon ignored. These methods are based on the so-called Secant equation. In our project we focus on solving a critical subproblem of the Quasi-Newton algorithm that requires determining a proper, suitable step size that takes from the current approximation to the minimum to a new 'better' one. The subproblem can either be posed as doing a Line Search along some generated search direction in order to determine a minimum along the search vector. Another technique, on which we focus primarily in this work, is to use a Trust Region method that directly computes the step vector without doing a focused Line Search. The subproblem is critical to the numerical success of Q-N methods. We emphasize features of successful implementation to pinpoint assess merits of Trust Region methods. Our Numerical Results reveal that Trust Region algorithms seem to markedly improve as the dimension of the problem increases, while for small dimensional problems performance of both methods is comparable.

PREFACE

Optimization might be defined as the science of determining the **best** solutions to certain mathematically defined problems, which are often models of physical reality. It involves the study of optimality criteria for problems, the determination of algroihtmic methods of solution, the study of the structure of such methods, and computer experimentation with methods both under trial conditions and on real life problems. There is an extremely diverse range of practical applications. Yet the subject can be studied (not here) as a branch of pure mathematics.

Before 1940 relatively little was known about methods for numercial optimization of functions of many variables. There had been some least squares calculations carried out, and Steepest Descent type methods had been applied in some Physics problems. The Newton method in many variables was known, and more sophisticated methods were being attempted such as the self-consistent field method for variational problems in theoretical chemistry. Nonetheless anything of any complexity demanded armies of assistants operating desk calculating machines. There is no doubt therefore that the advent of the computer was paramount in the development of optimization methods and indeed in the whole of numerical analysis and coupling those with AI techniques for globalizing the method. The 1940s and 1950s saw the introduction and development of the very important branch of the subject known as linear programming. (The term **programming** by the way is synonymous with **optimization** and was originally used to mean optimization in the sense of optimal planning.) All these methods however had a fairly restricted range of application [3].

The applicability of optimization methods is widespread; reaching into almost every activity in which numerical information is processed (Science, Engineering, Mathematics, Economics, Commerce, etc.). To provide a comprehensive account of all these applications would therefore be unrealistic, but a selection might include:

(a) Chemical reactor design;

(b) Aero-engine or Aero-frame design;

(c) Structure design-buildings, bridges, etc.;

(d) Commerce-resource allocation, scheduling, blending;

and applications to other branches of numerical analysis:

(e) Data fitting;

(f) Variational principles in p.d.e.s;

(g) Nonlinear equations in o.d.e.s.

The material of this project will be organized into Five Chapters. Chapter One will address the problem we are seeking the solution for. Chapter two describes Newton, and Quasi Newton methods and a comparison between them. In chapter three we will concentrate on the early Line Search methods. Chapter four is about Trust Region methods. Chapter Five is dedicated for Numerical Results and Conclusion.

CHAPTER I

INTRODUCTION

Optimisation theory is a byproduct of linear Algebra and Analysis. As people tried to understand, model and control large systems, new optimisation problems have arisen in economics, science, engineering, management, and technology. The prospect of solving such problems has only been realised in this century with the advent of computer algorithms.

Before talking in details about the minimisation problem, we present a number of definitions, which will referred throughout this project.

1.1 DEFINITIONS AND TERMINOLOGY

Please refer to the "symbols, Notation and Abbreviation" table for more details on the notation used in the following definitions.

Def. 1.1: (continuity)

A function F: $D \subseteq R^n \rightarrow R^m$ is continuous at $x' \in D$ if only if

$$\| F(x) - F(x') \|_m \rightarrow 0$$

whenever

$$\|x - x'\|_n \rightarrow 0$$

where $\|.\|_m$ is a norm on R^m and $\|.\|_n$ is a norm on R^n.

Def. 1.2: (unconstrained strict global minimum)

A Point x min is said to be a strict global minimum point of a function R^n if $f(x) > f(x_{min})$ for all $x \in R^n, x <> x_{min}$.

Def. 1.3: (strict local minimum)

A point x_{min} is said to be a strict local minimum of f over $\varphi \subseteq R^n$ if there is a $\tau > 0$ such that $f(x) > f(x_{min})$ for all $x \in \varphi$ with $\|x - x_{min}\| \leq \tau$ (for some norm $\|.\|$ and $x <> x_{min}$).

Def. 1.4: (positive definite matrix)

Asymmetric matrix A ($\in R^{nxn}$) said to be positive definite if, for all nonzero s $\in R^n$, $s^T As > 0$.

Def. 1.5: (convergence)

The sequence $\{x_i\}$ in R^n converges to $x_{sol} \in R^n$ if and only if

$$\lim_{i \to \infty} \|x_i - x_{sol}\| = 0,$$

for some norm $\|.\|$ on R^n.

Def. 1.6: (quadratic termination)

An iterative method that minimises an arbitrary quadratic function possessing an unconstrained strict global minimum in a finite number of iterations is said to possess "quadratic termination".

Def. 1.7: (directional derivative and curvature)

For any direction $p \in R^n$

$$x(t) = x + tp, \text{ where } t \in R^n \text{ and } x \in R^n,$$

the directional derivative of $f(x(t))$ along the line at any point x(t) is given by

$$\partial f / \partial t = p^T g(x(t)),$$

where $g(x(t))$ is the gradient of f evaluated at $x(t)$.

Likewise, the curvature or f along the line is

$$\partial^2 f / \partial t^2 = p^T G(x(t)) p,$$

where $G(x(t))$ is the Hessian of f evaluated at $x(t)$.

Def. 1.8: (descent direction)

p is a descent direction from x for a given function if the directional derivative of the function at x in this direction is negative.

$$p^T g(x(t)) < 0$$

Lemma 1.1

Let x and y be any two vectors in R^n, and let A be any $n*n$ real, symmetric positive definite matrix. The following inequality is true:

$$(x^T y)^2 \leq (x^T Ax)(y^T A^{-1} y).$$

1.2 THE PROBLEM

We focus attention here on problems of unconstrained optimisation over the whole vector space $\Re^n$. The problem we will consider throughout is that of finding an unconstrained local minimum, i.e..

$$\min f(x),$$

$$\text{where } f: \ \Re^n \rightarrow \Re.$$

A common approach to the construction of algorithms for the solution of minimization problems is as follows: Given $f(x)$ and an estimate of the minimum, x_0 , produce a sequence $\{x_i\} \in \Re^n$ that converges to x_{min}. An algorithm for producing such a sequence from an initial estimate x_0 is obtained by specifying a rule for producing a new approximation to the local minimiser based upon information obtained from the current approximation (such as the first partial derivatives and/or the second partial derivatives).

A global minimum can be numerically difficult to find, especially if the function is non-smooth or highly non-quadratic. In the course of locating a global minimum, several local minima could be encountered and the particular algorithm could halt at any of them.

Many practical problems are of the constrained type in that the permissible values for the variable x are confined to lie in some set. It is possible in some cases to convert a constrained problem into an unconstrained one [3]. The structure of most constrained optimisation problems is essentially contained in the following:

$$\min f(x)$$

$$\text{subject to}$$

$$C_k(x) = 0, \ k \in E,$$

$$C_j(x) \geq 0, \ j \in I,$$

where E and I are the index sets of equality and inequality constraints, respectively. Constrained optimization problems fall essentially into one of the two categories:

(I) Linearly-constrained programming

and

(II) Non-linear programming.

Category (I) tends to give rise to simpler problems in that, in many cases, the objective function is either linear or quadratic (linear or quadratic programming), and such problems can be solved in a finite number of steps. Category (II) is the more difficult since it involves non-linear constraints. If the constraints cannot be eliminated directly by reformulation, then one approach is to attempt converting the problem into a single unconstrained one or a sequence of such problems. For instance, one method of doing so is known as the Penalty Function method, in which a penalty term is added to the objective function for any violation of the constraints. Thus the problem becomes that of finding the unconstrained minimum of the function

$$z(x) = f(x) + rP(x),$$

Where $f(x)$ is the main objective function, r is a monotonically increasing scale factor which controls the effect of the constraint violation and $P(x)$ is the penalty function. The penalty function should be continuous on $\Re^n$ and should be zero if and only if x is in the feasible region. The function $z(x)$ in ,is minimized by performing unconstrained minimisation for a sequence of increasing values of r [4]. Ideally, as $r \to \infty$ the solution point of the penalty problem will converge to a solution of the constrained problem.

CHAPTER II

NEWTON AND QUASI-NEWTON METHODS

2.1 INTRODUCTION

This chapter starts with discussing the application of Newton's method to the solution of nonlinear equations. The practical difficulties of Newton's method are outlined and used in motivating Quasi-Newton methods.

Many of the existing optimization methods [and indeed methods for solving systems of non-linear equations] modify and build around Newton's method to create algorithms, which attempt to achieve its rate of convergence.

On the other hand, Newton's method requires demanding conditions to be successful. These limit its effectiveness in practice and, indeed, motivate the development of methods such as the Quasi-Newton methods.

2.2 NEWTON'S METHOD FOR SOLVING NONLINEAR EQUATIONS

The basic problem studied here is the solution of the system of nonlinear equations defined by means of the function

$$F : R^n \rightarrow R^n$$

For which a solution x_{sol} must be found such that $F(x_{sol}) = 0$. The function vector F is assumed to be continuously differentiable.

We use the following linear model to approximate F at $x_i + p_i$

$$m_i (x_i + p_i) = F(x_i) + J(x_i)p_i$$

where x_i is the current iterate and J ($\in R^{nxn}$) is the Jacobian matrix of F. This last model is used to determine the Newton step p_i, that takes us to the next iterate x_{i+1}, defined to be the zero of the model. Thus a typical iteration of the method is:

$$J(x_i)\, p_i = -F(x_i),$$

and

$$x_{i+1} = x_i + p_i$$

2.3 DRAWBACKS OF NEWTON

Newton's method possesses a very attractive local convergence rate, by comparison with other existing methods. However, the method suffers from implementation difficulties and its convergence is sometimes questionable due to the following drawbacks (even if p_i is used as a search direction along which a step is to be found such that the function value is reduced);

(a) Convergence to a saddle point is possible, due to an indefinite Hessian matrix.

(b) When G_i is singular, p_i is undefined.

(c) If G_i is not positive definite, p_i may not be a downhill direction.

(d) The need to derive and code expressions for the second derivative matrix, a process which is always susceptible to human error. It is this problem that motivates Quasi-Newton methods. This problem is, however, becoming less serious as automatic differentiation algorithms are being developed.

(e) The need to evaluate the Hessian Matrix at each iteration, a process which is often expensive. For this problem, several suggestions have been made of which we mention:

(a)- Evaluate the Hessian only at the first iteration. However it should be re-iterated that if evaluating the Hessian is an expensive process then coding it may be even more expensive at the first iteration, we then lose the method's speed of convergence

(b)- To evaluate the Hessian every m iterations (m is some positive integer). The above argument applies here as well with the difference that the method converges to the minimum with q-order $(m+1)^{1/m}$ [3]. This is more efficient than its predecessor is, especially if the Hessian is varying very little from iteration to iteration.

(c)- To obtain the matrix by finite differencing using first derivatives. This process is expensive in terms of gradient evaluations and usually requires n+1 gradient evaluations per iteration. Again, the speed of convergence is degraded especially if the matrix loses its symmetry and/or may not even positive definite.

(vi) The need to solve a linear system of equations at each iteration for p_i, which requires $O(n^3)$ operations. This problem can be avoided if the Hessian is in a factorized form. In this case, the problem of solving the system of linear equation is, thus, transformed to a backward and forward substitution.

(vii) There is the requirement to start close to the minimum inorder to guarantee convergence and also that the function must (in theory) have continuos derivatives of order 3. This leaves the possibility of divergence open.

2.4 QUASI-NEWTON METHODS

2.4.1 INTRODUCTION

Quasi-Newton methods are motivated by the drawbacks of Newton's method discussed in section 2.3 and, particularly, by the last three. The basic philosophy behind Quasi-Newton methods is to mimic Newton's method without requiring that the Hessian be available. Thus their most attractive feature is that implementing them requires only the first partial derivatives of the function to be available, the second derivative matrix is approximated. The approximation to the Hessian is built on a step-wise basis using only step and gradient information available at each iteration. The approximation matrices are usually constructed to satisfy the so-called Secant Equation (which will be derived in sections 2.4.2 and 2.4.3 for the cases of solving systems of non-linear equations and minimizing a function, respectively).

2.4.2 SOLVING NONLINEAR EQUATIONS

Again, we address here the problem

$$F(x) = 0, \text{ where } F: R^n \rightarrow R^n$$

We start by Newton's method for finding the step p_i from x_i to x_{i+1}.

In Quasi-Newton methods, the matrix $J(x_{i+1})$ is replaced by an approximate B_i, say, and, the transition function becomes:

$$B_i p_i = -F(x_i). \tag{2.1}$$

The next question is how to determine B_{i+1}, once x_{i+1} has been found. In order to determine a condition that the new Hessian approximation should satisfy, we build a Taylor series approximation of first order around x_{i+1} with B_{i+1} replacing G_{i+1} and require the relation to hold a an equality for $x = x_i$, as follows:

$$F(x_i) = F(x_{i+1}) + B_{i+1}(x_i - x_{i+1})$$

or, equivalently,

$$B_{i+1}p_i = y_i \tag{2.2}$$

Where

$$p_i = x_{i+1} - x_i \tag{2.3}$$

and

$$y_i = F(x_{i+1}) - F(x_i) \tag{2.4}$$

Equation (2.2) is widely known as the **Secant Equation** (or the "Quasi-Newton Equation"). It is obvious that equation (2.2) does not uniquely define the matrix B_{i+1}, since it involves n equations with n^2 unknowns and an infinite number of matrices may satisfy (2.2). [5]

An outline Quasi-Newton algorithm for solving non-linear equation is given by

ALGORITHM

1. Given F: $R^n \rightarrow R^n$, $x_o \in R^n$, $B_0 \in R^{nxn}$, i=0 (the iteration count) and the convergence tolerance ε.
2. Solve $B_i p_i = -F(x_i)$ for p_i.
3. Compute $x_{i+1} = x_i + p_i$, find $F(x_{i+1})$
4. Compute $y_i = F(x_{i+1}) - F(x_i)$
5. Use some updating formula, which satisfies (2.2), to obtain B_{i+1}.
6. If $||F(x_{i+1})|| > \varepsilon$ THEN $i = i + 1$, GOTO 2, ELSE stop.

2.4.3. UNCONSTRAINED OPTIMISATION

The Secant Equation is now intended for solving the problem.

$$\min_{x \in R^n} \quad f(x), \text{ where } f: R^n \rightarrow R.$$

For minimization problems, g_i is the gradient of f at x_i and, consequently, the matrix B_i is intended to approximate the Hessian (G_i). We are thus seeking to construct a Hessian approximation, which satisfies (2.2), with y_i in (2.4) replaced by

$$y_i = g_{(i+1)} - g_i \tag{2.5}$$

the direction vector p_i is obtained by solving

$$B_i p_i = -g_i.$$

By analogy with the Hessian G_i, B_{i+1} must be symmetric and, thus, for n equations, there are $(n^2 + n)/2$ unknowns in B_{i+1}, as opposed to n^2 unknowns for the unsymmetrical case of the Jacobian approximation. We now show that if the Hessian

$$p_i^T g_i = -p_i^T H_i p_i < 0.$$

approximation is positive- definite, then the quasi- Newton direction is downhill, for

$$p_i = -H_i g_i.$$

In this case, the direction vector is computed using

We note here that, as suggested for Newton's method, the direction vector p may be used to find a step size α such that

$$f(x_i + \alpha p_i) < f(x_i)$$

Since setting the step size, α, to one may not result in a sufficient reduction in the function value. In the case p is used as a search direction, p in (2.2) is replaced by αp_i (s_i).

The algorithm can be outlined as follows:

ALGORITHM I

1. **Given** x_o, choose some positive-definite metrix B_0 [or H_0] and set $i = 0$ (the iteration count).

2. **Compute** p_i, using $B_i p_i = -g_i$ (or $p_i = -H_i g_i$).

3. **Compute** the step vector s_i using either

(a) A Line Search technique to solve

$$t_i = \underset{t \in R}{arg\ min} f(x_i + \alpha p_i),$$

so that $s_i = t_i p_i$

or

(b) A Trust Region method to solve

$$min\ f(x_i + s_i) \quad \text{subject to } \|s_i\| \leq \delta_i,$$

where $\delta_i \in R$ is a positive scalar that defines the radius of some ball in which the quadratic model approximation of $f(x_i + s_i)$ is trusted to be acceptable [for details, see chapter four].

4. **Compute** $x_{i+1} = x_i + s_i$ and $g(x_{i+1})$.

5. **Compute** y_i using equation (2.5).

6. **IF** (i=0) and (the problem dimension is "large"), scale H_i or B_i by a suitable constant.

7. **Compute** H_{i+1} or B_{i+1} using updating formulas so that (2.2) [or some variant] is satisfied [with s_i replacing p_i] (see section 2.5) .

8. **IF** "termination criteria" (see next section) are met then **STOP** else $i = i + 1$, **GOTO 2**.

In step (1), the unit matrix is usually used as the initial approximation to the Hessian.

2.4.4 TERMINATION CRITERIA

The algorithms outlined above are usually halted on the basis of one or more of the following tests:

(a) The gradient test:

A test on the gradient size is conducted. This has the form $\|g\| \leq \varepsilon$, for some small positive $\varepsilon \in R$. The quantity ε specifies an upper limit on the gradient size. Ideally, $\|g\| = 0$ is what is desired but finite precision arithmetic renders such a test almost impossible to satisfy.

(b) The function value test:

$| f_{i+1} - f_i | < \varepsilon$, for some small positive ε.

(c) The step norm test:

Checks if the progress made between the last two iterates is insignificant:

$$\| x_{i+1} - x_i \|_2 < \varepsilon.$$

(d) The step components test:

This test is conducted component-wise on the two last iterates

$$|x^j_{i+1} - x^j_i | < \varepsilon, \text{ j} = 1.....\text{n}.$$

(e) Testing the predicted change in function value [given as the difference between the current function value and that obtained from an approximation to $f(x_i + p_i)$, evaluated at $[x_{i+1}]$:

$$\tfrac{1}{2} g_i^T H_i g_i \leq \varepsilon \text{ (for Q-N methods)}$$

or

$$\tfrac{1}{2} g_i^T G_i^{-1} g_i \leq \varepsilon \text{ (for Newton's method).}$$

(f) The iteration count test: for this test the iteration count is given a pre-specified upper limit, Tests (c), (d) and (e) are useful in that they indicate that the matrix H_i (or G_i^{-1}) or B_i may have become singular. Singularity of the matrices may be the cause of two iterates being close to each other's. The consequence is that the algorithm may fail to reach the minimum and will keep iterating in a restricted subspace. However, these last three tests may result in premature termination. Test (a) does not suffer from this disadvantage even though it may cause convergence to a saddle point. Test (a), combined with any or none of the other tests, is usually sufficient for testing convergence to either a saddle point or the minimum.

2.5 DERIVATION OF THE UPDATING FORMULA

The aim now is to define criteria for deriving the updating formula B_{i+1} (or H_{i+1}) to the Hessian approximation (or its inverse). These approximations satisfy the Secant Equation $B_{I+1}s_i=y_i$. In general such formula will have the form

$$H_{i+1}=H_i+U_H \tag{2.6}$$

for the inverse Hessian approximation and

$$B_{i+1}=B_i+U_B \tag{2.7}$$

for the Hessian approximation

In (2.6) U_H must satisfy

$$U_H y_i=s_i-H_i y_i$$

while for (2.7) we require

$$U_B s_i=y_i-B_i s_i$$

where $s_i=x_{i+1}-x_i$ and $y_i=g(x_{i+1})-g(x_i)$. The correction matrix U_H can be chosen to be a low rank matrix in order not to corrupt the information accumulated in H_i from previous iteration. Another reason is to be able to obtain the inverse of U_H (or U_B).

One of the most important updating formula that proved itself a serious contender in practice is known as the BFGS method, derived independently by Broyden, Fletcher, Goldfarb and Shanno. It is given by

$$U_H=\left(1+\frac{y_i^T H_i y_i}{s_i^T y_i}\right)\frac{s_i s_i^T}{s_i^T y_i}-\frac{s_i y_i^T H_i+H_i y_i s_i^T}{s_i^T y_i}$$

The superiority of this method in practice is achieved particularly when the Line Search is not exact [5].

Some of the problems that will safeguard the Newton like methods, and consequently producing a robust optimization algorithm, are the Trust Region and / or Line Search, which are sub-problems, that should be solved at each iteration when finding the minimum of $f(x)$. For Newton-like methods, instead of only using the search direction as a direct step from one iterate to another, the direction vector is used in determining a step vector from the current iterate to the next one by solving special problems that we will address in details in the next chapter.

CHAPTER III
LINE SEARCH

3.1 DESCENT METHODS AND STABILITY

Line search methods have frequently been used as means of introducing a degree of reliability and stability into optimization software and this idea is followed up in this section. The Line Search method is tailored for solving the problem

$$\min_{t \in R} f(x_i + tp_i) \text{ s.t } t>0$$

This method is designed to safeguard Newton-Like methods and, thus, to produce robust optimization algorithms. As appropriate, the sub-problem has to be solved at each iteration in order to reach the minimum. In the early days one common strategy was to choose the step $t_{(i)}$ close to the value given by an exact Line Search. This is motivated by early theory, which shows that the Steepest Descent method with an exact Line Search is globally convergent to a stationary point [1]. However accurate Line Searches are expensive to carry out, and there is also the nuisance that the exact minimizer may not exist. Other researches weakened the Line Search tolerance considerably and used the Descent property merely to force a decrease $f_{(i+1)} < f_{(i)}$ in the objective function on each iteration. This usually turned out to be more efficient. However merely requiring a decrease in f does not ensure global convergence so there were doubts about the stability of this more efficient approach. This fact, and the proliferation of numerous different codes for the Line Search, led to the development of a generally accepted set of conditions for terminating the Line Search which would allow accuracy Line Searches while forcing global convergence. [See section 3.2].

LINE SEARCH algorithms can generally be outlined as:

Algorithm 3.1

1. Given p_i, g_i and f_i; set k = 0 and $t_i = t_k^{(i)}$ (an estimate of the line minimum along p_i);
2. Use the available information [listed in step (1) above] to locate a better estimate $t_{k+1}^{(i)}$ of the line minimum using some method (for e.g., Cubic Interpolation).
3. SET $t_i = t_{k+1}^{(i)}$, $g_{i+1} = g(x(t_i))$, $f_{i+1} = f(x_i + t_i p_i)$; increment k.
4. **GOTO 2** unless **converged**

In practice, of course, it is impossible to obtain the exact minimum point called for by the ideal Line Search algorithm. As a matter of fact, it is often desirable to sacrifice accuracy in the Line Search routine in order to conserve overall computation time. Because of these factors we must, to be realistic, be certain, at every stage of development, that our theory does not crumble if inaccurate Line Searches are introduced.

Inaccuracy generally is introduced in a Line Search algorithm by simply terminating the search procedure before it has converged exactly.

In the next section we present some commonly used criteria for terminating a Line Search.

3.2 TERMINATION CRITERIA

3.2.1 Percentage test

One important inaccurate Line Search algorithm is the one that determines the search parameter t to within a fixed percentage of its true value. Specifically, a constant c, $0<c<1$ is selected ($c = 0.10$ is reasonable) and the parameter t in the Line Search is found so as to satisfy $|t - t'| \leq ct'$ where t' is the true minimizing value of the parameter.

3.2.2 Armijo test

A practical and popular criterion for terminating a Line Search is Armijo's rule. The essential idea is that the rule should first guarantee that the selected t is not too large, and next it should not be too small. Let us define the function

$$f(t) = f(x_i + tp_i)$$

Armijo's rule is implemented by consideration of the function $f(0) + \varepsilon f'(0)t$ for fixed ε, $0<\varepsilon<1$. This function is shown is shown in Fig. III(a) as the dashed line. A value of t is considered to be not too large if the corresponding function value lies below the dashed line; that is, if

$$f(t) \leq f(0) + \varepsilon f'(0)t. \qquad (3.1)$$

To insure that t is not too small, a value $\eta>1$ is selected, and t is then considered to be not too small if

$$f(\eta t) > f(0) + \varepsilon f'(0)\eta t.$$

This means that if t is increased by the factor η, it will fail to meet the test (3.1). The acceptable region defined by the Armijo rule is shown in Fig.III (a) when $\eta = 2$.

Sometimes in practice, the Armijo test is used to define a simplified Line Search technique that does not employ curve-fitting methods. One begins with an arbitrary t. If it satisfies (3.1), it is repeatedly increased by η($\eta = 2$ or $\eta= 10$ and $\varepsilon = 2$ are the often used) until (3.1) is not satisfied, and then the penultimate t is selected. If, on the other hand, the original t does not satisfy (3.1), it is repeatedly divided by η until the resulting t does satisfy (3.1).

3.2.3 Goldstein's test

It essentially uses test (3.1) but with $0 < \varepsilon < ½$ and replaces the second test by

$$f(t) > f(0) + (1- \varepsilon)\, t f'(0).$$

Thus Goldstein's parameter t must satisfy

$$\varepsilon \leq \frac{f_{i+1} - f_i}{t p_i^T g_i} \leq 1 - \varepsilon$$

3.2.4 Wolfe Test

If derivatives of the objective function, as well as its values, can be evaluated relatively easily, then the Wolfe test, which is a variation of the above, is sometimes preferred. In this case ε is selected with $0<\varepsilon<½$, and t is required to satisfy (3.1) and

$$f'(t) \geq (1 - \varepsilon) f'(0).$$

This test is illustrated in Fig.III(c). An advantage of this test is that this last criterion is invariant to scale-factor changes, where as in the Goldstein test is not. Another test, which we have adopted in our experiments, also uses (3.1) in addition to

$$f'(t) > \beta f'(0) \qquad (3.2)$$

{Obtained by rewriting Wolfe's test}. Condition (3.2) requires that the rate of change of f in the direction p at x_i (a_i) be larger than some prescribed fraction of the rate of decrease in the direction p at $x(0)$.

The constants in both tests must satisfy:

$$\varepsilon \in (0,0.5] \text{ and } \beta \in (\varepsilon, 1].$$

3.3 FIBONACCI SEARCH

A very popular method for resolving the Line search problem is the Fibonacci search method described in this section. The method has a certain degree of theoretical elegance, which no doubt partially accounts for its popularity, but on the whole, as we shall see, there are other procedures, which in most circumstances are superior.

The method determines the minimum value of a function f over a closed interval [c1, c2]. In applications, f may in fact be defined over a broader domain, but for this method a fixed interval of search must be specified. The only property that is assumed of f is that it is unimodal, that is, it has a single relative minimum (see Fig. III(d)) The minimum point of f is to be determined, at least approximately, by measuring the value of f at a certain number of points. It should be imagined, as is indeed the case in the setting of nonlinear programming, that each measurement of f is somewhat costly of time if nothing more.

To develop and appropriate search strategy, that is, a strategy for selecting measurement points based on the previously obtained values, we pose the following problem. Find how to successively select N measurement points so that, without explicit knowledge of f, we can determine the smallest possible region of uncertainty in which the minimum must lie. In this problem the region of uncertainty is determined in any particular case by the relative values of the measured points in conjunction with out assumption that f is unimodal. Thus, after values are known at N points $x_1, x_2, \ldots, x_N$ with

$$c_1 \le x_1 < x_2 \ldots < x_{N-1} < x_N \le c_2,$$

the region of uncertainty is the interval $[x_{k-1}, x_{k+1}]$ where the minimum point is among the N, and we define $x_0=c_1$, $x_{N+1}=c_2$ for consistency. The minimum of f must lie somewhere in this interval.

The derivation of the optimal strategy for successively selecting measurement points to obtain the smallest region of uncertainty is fairly straightforward but somewhat tedious. We simply state the result and give and example.

Let

$d_1 = c_2 - c_1$, the initial width of uncertainty

d_k = width of uncertainty after k measurements. (Decreasing)

Then, if the total of N measurements is to be made, we have $d_k < d_{k-1}$

$$d_k = \left(\frac{F_{N-k+1}}{F_N}\right) d_1,$$

where the integers F_k are members of the Fibonacci sequence generated by the recurrence relation

$$F_N = F_{N-1} + F_{N-2}, \qquad F_0 = F_1 = 1.$$

The resulting sequence is 1, 1, 2, 3, 5, 8, 13,....

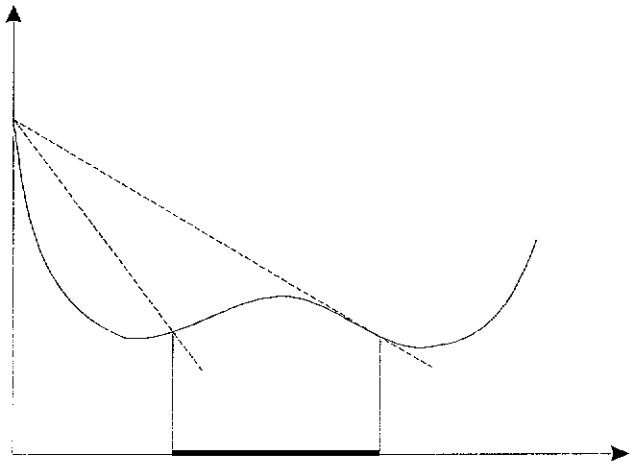
The procedure for reducing the width of uncertainty to d_N is this: The first two measurements are made symmetrically at a distance of $(F_{N-1}/F_N)d_1$ from the ends of the initial intervals; according to which of these is of lesser value, and uncertainty interval of width $d_2=(F_{N-1}/F_N)d_1$ is determined. The third measurement point is placed symmetrically in this new interval of uncertainty with respect to the measurement already in the interval. The result of this third measurement gives and interval of uncertainty $d_3 = (F_{N-2}/F_N)d_1$. In general, each successive measurement point is placed in the current interval of uncertainty symmetrically with the point already existing in that interval.

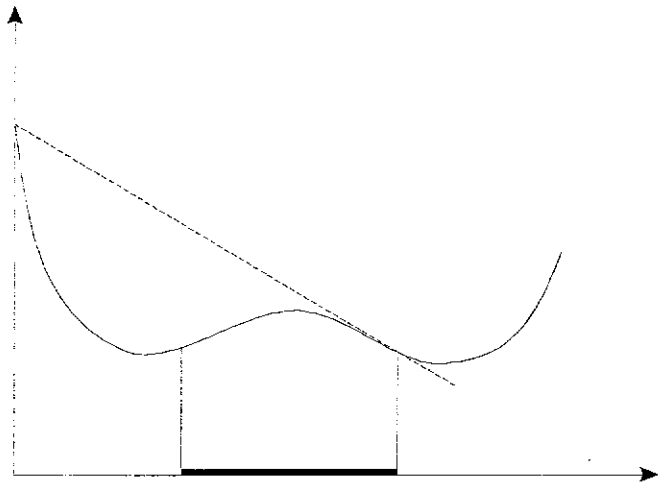
3.4 CUBIC INTERPOLATION

This Line search algorithm constructs a cubic polynomial that interpolates the data. $f(t_L)$, $f(t_U)$, $f'(t_L)$, and $f'(t_U)$, where t_L and t_U are the extreme points of some initial search interval $[t_L,t_U]$. The minimum point of the cubic curve and the search interval size changes estimates the line minimum as one of the extreme points is replaced by the estimated minimum (on the basis of some tests). The process is carried out until conditions such as (3.1) and (3.2) are satisfied.

A general frame of Cubic Interpolation algorithm is as follows:

1. Given t_L (=0), $f(t_L)$, $f'(t_L)$, t_U, $f(t_U)$, and $f'(t_U)$;
2. If test (3.1) succeeds (with $t = t_U$) then
   {if (3.2) succeeds return t_U;
   else
   {/*t_U is too small*/
   $t_L = t_U$; $f(t_L) = f(t_U)$; $f'(t_L) = f'(t_U)$;
   $t_U = wt_U$ (w >1); find $f(t_U)$ and $f'(t_U)$;
   go to 2;}
   }
3. else /* if (3.1) fails*/
   {a- do a cubic fit in the available points
   and determine t_C (the minimum of the cubic), $f(t_C)$,
   and $f'(t_C)$;
   b- if test (3.1) succeed (with $t = t_C$) then
   {if condition (3.2) succeeds (with $t = t_C$) return t_C
   else
   {$t_L = t_C$; $f(t_L) = f(t_C)$; $f'(t_L) = f'(t_C)$; go to 3a;}
   }
   else /* t_U is too large */
   {$t_U = t_C$; $f(t_U) = f(t_C)$; $f'(t_U) = f'(t_C)$; go to 3a;}
   }





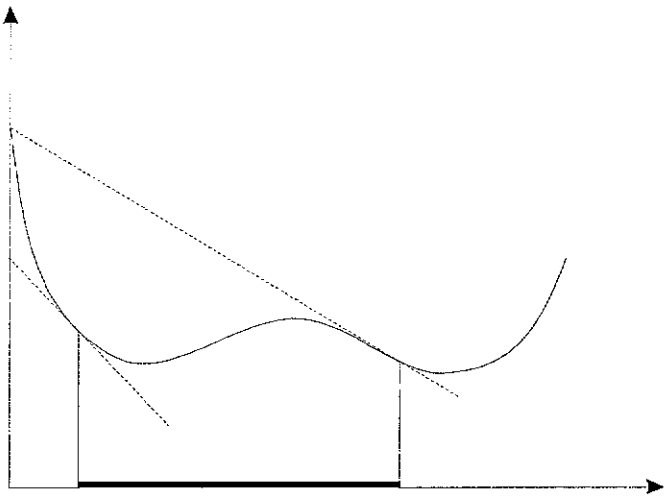


Fig III

(a) Armijo Rule.

(b) Golestein rule

(c) Wolfe test

CHAPTER IV
TRUST REGION

4.1 INTRODUCTION

When one has a poor approximation to an unconstrained minimiser, quick improvement is likely to result from taking a step a long the negative gradient. This approach bogs down as the minimum is approached. But, when a good approximation minimiser is available, the Quasi-Newton methods improve it quickly to acceptable accuracy.

Powell introduced a strategy in an unconstrained minimisation which can be viewed essentially as an elegant way to change from the gradient to the Q-N steps as the approximate minimiser is improved. Powell's strategy can be geometrically seen as a line joining x_i, and the negative gradient, and then sliding from the top of the negative gradient to the top of the Q-N to reach our target which is a suitable step (s_i). From this s_i we can have the starting point of the next iteration. $x_{i+1} = x_i + s_i$.

Powell's strategy (Dogleg) seeks to

$$\text{Minimize } \varphi(s_i) \quad \textit{subject to } \| s \| \leq \delta_i \qquad (4.1)$$

$$s \in R^n$$

, where

$$\varphi(s_i) = f(x_i) + s_i g_i^T + \tfrac{1}{2} s_i^T B_i s_i \qquad (4.2)$$

A useful advice is to think of $x_i + s_i$ as a region in which we can trust $\varphi(s_i)$. The size of this region can be varied depending on the trustworthiness of (4.2). If the predicted function value is good, s_i can be increased and, if the prediction is poor, s_i can be decreased.

So during one iteration of a full optimising a Trust Region algorithm will take the following steps:

Algorithm II (Trust Region Algorithm).

(For algorithm I see section 2.4.3).

(i) Given x_i, f_i, g_i and B_i find the step s_i which minimizes $\varphi(s_i)$ subject to $\| s_i \| \leq \delta_i$. (Algorithm III).

(ii) Decide if this Step s_i is acceptable (according to conditions explained in section 4.4). If not, adjust the size of the Trust Region and return to (i). (Algorithm IV).

4.2 The Dogleg Method

In order to find x_{i+1} Powell first calculate the Cauchy point, i.e., the point x_i-$\alpha^* g_i$ where

$$\alpha^* = \frac{g_i^T g_i}{g_i^T B_i g_i}, \tag{4.3}$$

which minimises $\varphi(-\alpha^* g_i)$. If the Cauchy point is outside the circle center at x_i, with redius δ_i then

$$x_{i+1} = x_i - \beta\alpha^* g_i$$

where β is chosen so that

$$\| \beta\alpha^* g_i \| = \delta_i$$

The reasoning is that the gradient is still large, so we should continue to move along it. If the Cauchy point is inside the Trust Region, then we assume that it is the time to calculate the Newton point $x_i - H_i g_i$. If the Newton point is inside the circle, we pick

$$x_{i+1} = x_i - H_i g_i ;$$

otherwise we pick

$$x_{i+1} = x_i + d_i$$

where

$$\| d_i \| = \delta_i$$

$$d_i = (1-\theta_i)(-\alpha^* g_i) + \theta_i(-H_i g_i), \quad \text{with } 0 \leq \theta_i \leq 1. \tag{4.4}$$

4.3 Double Dogleg Method

One of the reasons why people use Powell's dogleg method is that it has excellent global convergence properties. However, when we are fairly close to the solution, a Dogleg step seems to have more bias toward the gradient direction and the result is apparently slower convergence. Dogleg method is modified so that more bias toward the Newton step is introduced[2].

$$s^{CP} = -\frac{\| g_i \|^2}{g_i^T B_i g_i} g_i,$$

Let s^{CP} be the Cauchy step, s^N be the full Q-N step,$s^{\hat{N}}$ is the reduced Q-N step i.e.,

$$s^N = -H_i g_i$$

In fact the reduced Q-N step lies in the Q-N direction but variable within a certain bound η that will be discussed later

When

$$s^{CP} < \delta_i \text{ and } s^N > \delta_i$$

we first look for the point in the Newton direction at which the reduction of the quadratic approximation to f resulting from taking this step is the same as taking the Cauchy step.

That is, we determine η such that

$$f_i(-\eta H_i g_i) = f(s^{CP}) = f_i\left[\frac{-\| g_i \|^2}{g_i^T B_i g_i}\right] g_i,$$

which yields after equating the two formulas

$$\eta^2(\tfrac{1}{2} g_i^T H_i g_i) - \eta(g_i^T H_i g_i) + \tfrac{1}{2}\left[\frac{-\| g_i \|^4}{g_i^T B_i g_i}\right] = 0$$

where

$$\eta = \frac{g_i^T H_i g_i \pm \sqrt{(g_i^T H_i g_i)^2 - (\frac{\| g_i \|^2 \, g_i^T H_i g_i}{g_i^T B_i g_i})}}{g_i^T H_i g_i}$$

$$= 1 \pm \sqrt{\frac{1 - \| g_i \|^4}{g_i^T H_i g_i g_i^T B_i g_i}} = 1 \pm \sqrt{1-c},$$

and

$$c = \sqrt{\frac{1 - \| g_i \|^4}{g_i^T H_i g_i g_i^T B_i g_i}}$$

It can be shown that $c \leq 1$ and η is well defined.

We then take the dogleg step between s^{CP} *and* ηs^N *if* $\|\eta s^N\| > \delta_i$, in practice, it seems best to take η as

$$\eta = 1 - \alpha\sqrt{1-c},$$

Where α is some positive number less than 1, which we used it in our implementation as,

$$\alpha = 0.8\sqrt{1-c}$$

from the stated above $\eta=0.8c+0.2$.

In the dogleg curve we must move from the Cauchy point to $\tilde{N}$ with φ decreasing monotonically.

Now that we have fully described the double Dogleg curve we can define an algorithm to calculate s_i. Clearly, we wish to ascertain if $\| s^N \| < \delta_i$ and, if not, find the point where the double Dogleg curve crosses the boundary of the Trust Region. This involves only four checks.

1) If $\| s^N \| < \delta_i$ then $s_i = s^N$ (see Fig. 4.1).

2) Next check if $\| s^{\tilde{N}} \| \leq \delta_i$ (i.e. if is in the Trust Region) ; if it is then so is the Cauchy Point (see Fig. 4.2). We deduce that a point on the curve between $\tilde{N}$ and N is where the curve with the Trust Region boundary intersect; then we take a step of length δ_i in the Quasi- Newton direction.

3)If $\|s^{\tilde{N}}\| > \delta_i$ then we check if $\| s^{CP} \| > \delta_i$. If so (see Fig. 4.3) then a point on the curve between x_i and *C.P.* is where the Trust Region boundary and curve intersect and we take a step of length δ_i in the direction of Steepest Descent.

4)If none of these cases is satisfied then we are left with the boundary of the Trust Region intersection with the curve at a point between *C.P.* and $\tilde{N}$ i.e. we have a step direction somewhere between the direction of steepest decent and the Quasi-Newton direction. In this case we must calculate the λ for which:

$$\|x_i + s^{CP} + \lambda(\eta s^N - s^{CP})\| = \|x_i + s_i\| = \delta_i \quad (4.6)$$

$$\text{Let } s^{\tilde{N}CP} = \eta s^N - s^{CP} \quad (4.7)$$

and

$$\alpha = (s^{\tilde{N}CP})^T s^{CP}; \quad \beta = (s^{\tilde{N}CP})^T (s^{\tilde{N}CP}); \quad \gamma = \|s^{CP}\|$$

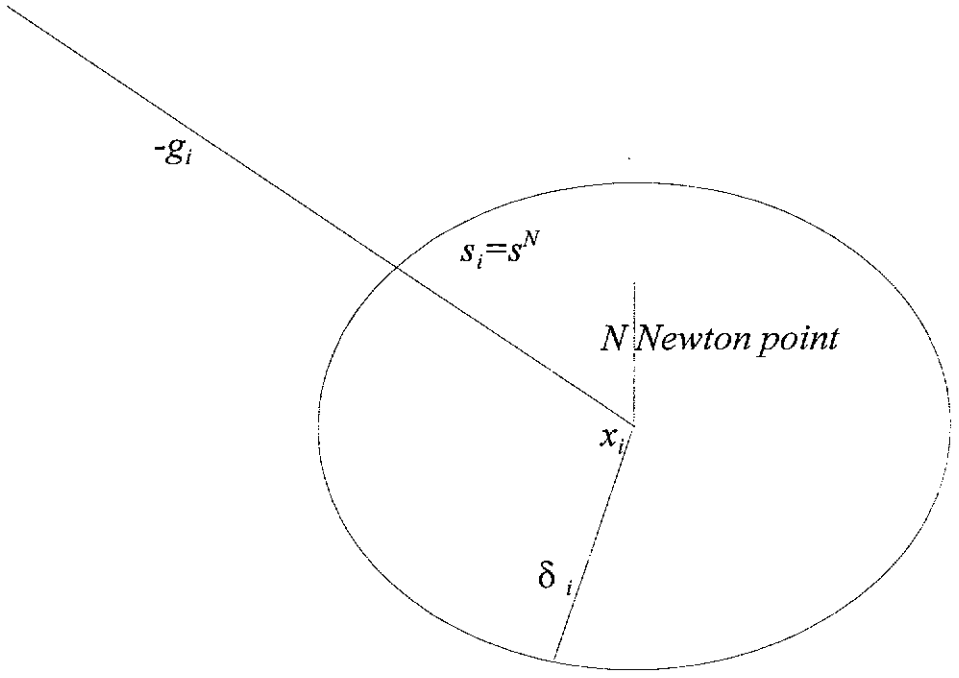


FIGURE 4.1

$S^N < \delta_i$

Then we have

$$\lambda = \frac{\sqrt{-\alpha + \alpha^2 - \beta(\gamma^2 - \delta_i^2)}}{\beta} \quad (4.8)$$

So λ is easily calculated from current information and we set $s_i = s^{CP} + \lambda(\eta s^N - s^{CP})$

We now have Algorithm IV to perform Step (i) of Algorithm II:

Algorithm IV (Dogleg Step Algorithm).

(i) Calculate s^{CP}, $s^{\tilde{N}}$ and s^{N} (only on first call of Algorithm IV main program iteration)

(ii) Check for possibility of a full Newton step:

If $s^{N} < \delta_i$ THEN RETURN $s_i = s^{N}$ to algorithm II.

ELSE If $s^{\tilde{N}} < \delta_i$ THEN RETURN $s_i = \dfrac{\delta_i s^{\tilde{N}}}{\| s^{\tilde{N}} \|}$ to Algorithm II .

ELSE IF $s^{CP} < \delta_i$ THEN RETURN $s_i = \dfrac{\delta_i s^{CP}}{\| s^{CP} \|}$ to Algorithm II .

ELSE calculate λ using (4.8) and RETURN $s_i = s^{CP} + \lambda (\eta s^{N} - s^{CP})$.

The Dogleg algorithm therefore involves four simple checks and these easy calculations after the Dogleg curve has been initially set up.

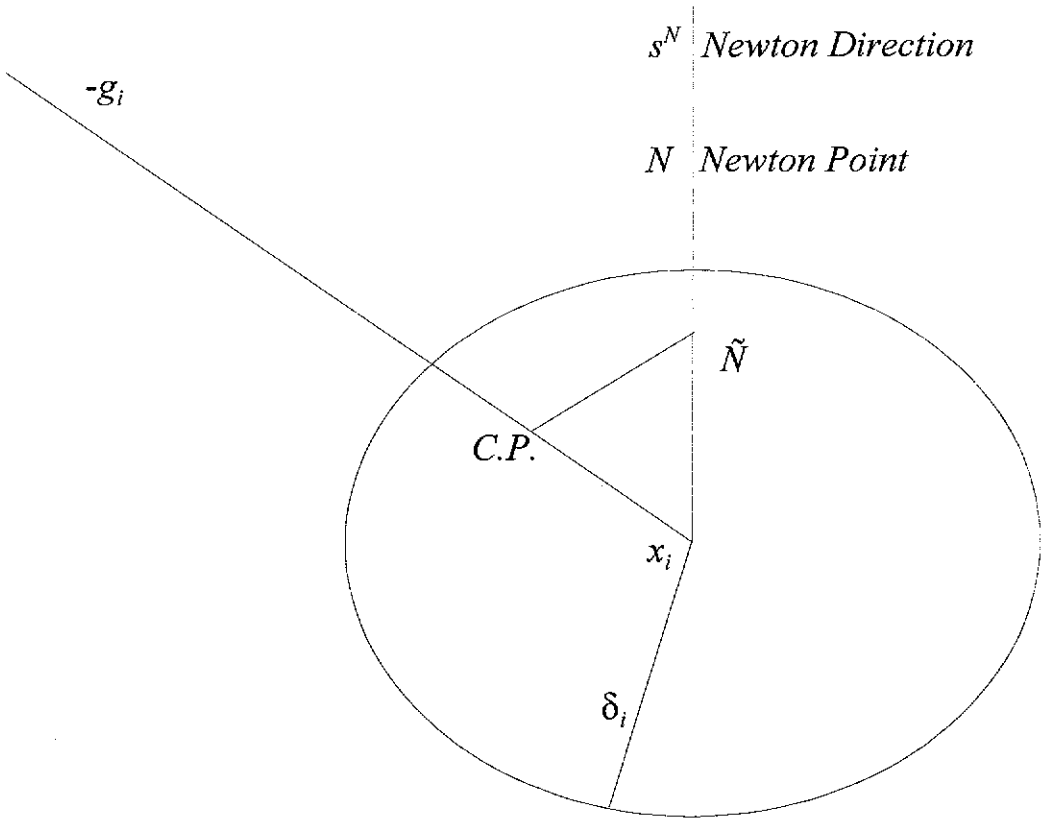


FIGURE 4.2

$s^{\tilde{N}} < \delta_i$

Referring to Algorithm II, we see that the step calculated in step (i) must be checked for acceptability in step (ii). Step (i) of Algorithm IV is only carried out on entry to Algorithm II and for one complete iteration of the main program the Dogleg curve remains fixed. Section 4.4 describes how δ_i is varied to decide on acceptability of s_i. Even if the first pass of Algorithm III renders no full Quasi-Newton step, it is possible that the size of the Trust Region can be increased and a full Quasi-Newton step finally accepted. So Algorithms IV excludes the possibility of choosing such a step even if the first attempt is unsuccessful. In contrast a full Quasi-Newton step may be unacceptable by Algorithm II Step (ii) and a smaller step may finally be chosen.

In the next section we discuss in more detail how such decisions may be taken.

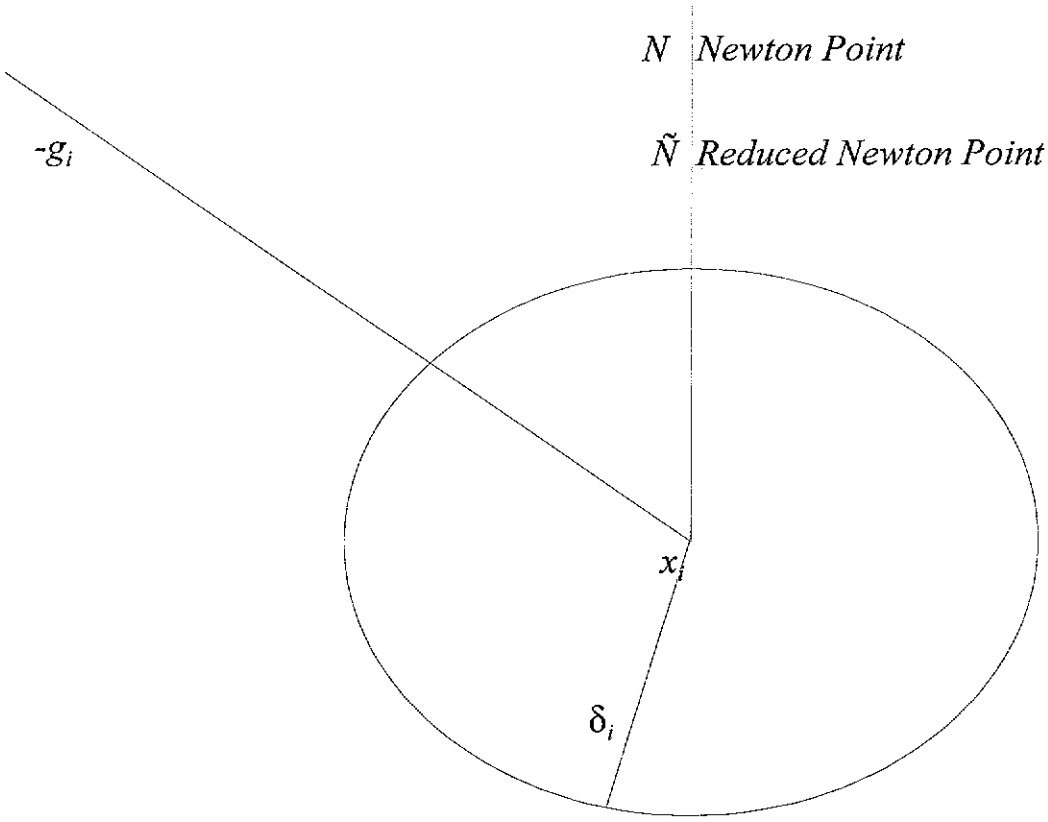


FIGURE 4.3

$s^{C.P} > \delta_i$

Where $s^{C.P.}$ is the step in Cauchy Point direction

4.4 ACCEPTANCE OF THE STEP AND VARYING THE TRUST REGION SIZE

We must now define some rules for the acceptance of the step s_i and for the varying of the size of the Trust Region if is unacceptable (Step (ii) of Algorithm II). The principle is that the Doglegs curve remains fixed from the first call of Algorithm II and is varied by Algorithm IV until Algorithms III produces an s_i which is acceptable. Naturally the major constraint is that at $x_{i+1}=x_i+ s_i$ the function value must be lower than at x_i i.e $f_{i+1}<f_i$. This condition will not always be met for instance, if δ_i is too large then the algorithm could shoot past the minimum to point where the function value is larger.

Thus to prevent $f_{i+1} \geq f_i$, we impose the restriction :

$$f_{i+1} \leq f_i + \alpha g_i^T s_i \qquad \alpha = 10^{-4} \tag{4.9}$$

since $\alpha g_i^T s_i < 0$ this implies $f_{i+1}<f_i$. If (4.9) is not satisfied then δ_i is too large and must be decreased to obtain a smaller f_+ when s_i is recalculated . To find the optimum size of δ_i we minimize f_i in the direction s_i from x_i and to do this we approximate $f(x_i+\lambda_* s_i)$ by a quadratic model which is minimized at

$$\lambda_{*=} \frac{-g^T s_i}{2[f_{i+1}-f_i- g^T_i s_i]} \tag{4.10}$$

The new δ (given by $\delta_+ = \|x_{i+1}-x_i\|$), which is obtained, is constrained to lie in $[0.1\delta_i, 0.5\delta_i]$. Thus δ_i shrinks by 50-90% and Step (i) of Algorithm II is performed again to calculate a smaller Step s_i which hopefully will enable 4.9 to be satisfied. If not the above process is carried out again. Of course the opposite can occur and the Trust Region size can be too small. Two cases have to be considered which indicate that δ_i is too small and s_i unacceptable.

The first is that (4.9) is satisfied but is satisfied much to well. If we have:

$$f_{i+1} \leq f_i + g_i^T s_i \leq f_i + \alpha g_i^T s_i \tag{4.11}$$

Then f has decreased in value , relatively speaking , by a very large amount and we can consider that it will carry on decreasing rapidly from x_{i+1} .This implies that a larger step can be taken.

The second case is one where the quadratic model (4.1) is predicting the function value much too well. A tolerance can be placed on the model performance thus:

$$| \Delta f_{\text{pred}} - \Delta f | \leq 0.1 | \Delta f | \tag{4.12}$$

where

$$\Delta f_{\text{pred}} = \Delta f_{\text{pred}\,(i+1)} - f_i$$

and

$$\Delta f = f_{i+1} - f_i$$

In other words, there is a percentage error of less than 10% between the quadratic model and the actual function values, then the model is performing "too well" and the Trust Region size can be increased.

Arbitrarily it is decided to double δ_i and retry step (i) of Algorithm II doubling of δ_i is performed until any of (4.9), (4.11), (4.12) are not satisfied, i.e. until doubling of δ_i is not justified. In the case of (4.9) not being satisfied we drop back to the previous x_{i+1}. There is also a chance that f_{i+1} is greater than the previous f_{i+1} before (4.9) fails and this case we also drop back to be previous step.

The above controls on s_i should finally yield a satisfactory step s_i and also adjust δ_i to a more suitable value. This value of δ_i should be a good starting point for the next iteration of the main program (Algorithm I), but we carry out one final check on δ_i before leaving Algorithm II to proceed with Step (iv) of Algorithm I.

If the model is predicting well, i.e.:

$$\Delta f \leq 0.75\, \Delta f_{\text{pred}}, \tag{4.13}$$

we double δ_i in preparation for the next call of Algorithm II.

If it is predicting badly, i.e.:

$$\Delta f \geq 0.75\, \Delta f_{\text{pred}} \tag{4.14}$$

We halve δ_i for the next iteration.

Otherwise δ_i is acceptable and is left unaltered .

So the basic steps for the above conditions are:

Algorithm IV (Step acceptance and Trust Region updating algorithm).

(a) IF δ_i was doubled in a previous call THEN

IF (4.9) is not satisfied OR $f_{i+1} \geq f_{\text{prev}\,(i+1)}$ THEN

drop back to previous step.

Step s_i is now acceptable (Step (iv) is not required).

RETURN to Algorithm I.

(ii) IF δ_i has not previously been doubled THEN

IF (4.9) is not satisfied THEN

Set $\delta_{i+1}=\lambda_* \|x_{i+1}-x_i\|$ ensuring that $\delta_{i+1} \in [0.1\delta_i, 0.5\delta_i]$

RETURN to Algorithm III.

(iii) IF δ_i has not previously been reduced THEN

IF one of (4.10) OR (4.11) is satisfied THEN

Set $\delta_{i+1}=2\delta_i$

RETURN to Algorithm III.

(iv) Step s_i is now acceptable .

Check conditions (4.13) and (4.14) and set δ_i accordingly

RETURN to Algorithm I.

By the conditions imposed, if the Trust Region starts to be increased it cannot be decreased and vice-versa.

CHAPTER V

NUMERICAL RESULTS AND CONCLUSION.

5.1 EXPERIMENTAL RESULTS AND ANALYSIS

In this Chapter we conduct the numerical results obtained from the experiments on 20 functions using both Trust Region Method (Double Dogleg) and the Line Search Method. We specify for each function the starting point we used in our tests. Each starting point corresponds to a different problem number, and this function number is the one referred to by our tabular results. (See Appendix I). Most of our functions are taken from the list presented by More, Garbow, and Hillstorm [5]. In our presentation of the starting point, we have used the notation "[...]*" to denote that the sequence in the bracket is repeated as many time as necessary to fill in the component of the vector x_0. The functions are listed with starting points numbered according to the problem number referred to by the tabular results. (See Appendix II)

The results summarised in this chapter are based on testing the problems for three-dimensional categories, as follows: as small as less than 16, then medium with dimension 40, and finally large with dimension that reaches 80.

We now explain the entries in the tables in Appendix II. We have used the abbreviation **NOI, FC,** and **GC** to refer to the total number of iterations, number of functions, and the number of gradient evaluations, respectively.

Our comparison of the performance of the tested methods will be carried out on the basis of function/gradient evaluations, which may be regarded as being the dominating factor in the optimization process. Ties are resolved using the iteration count. The method with * means that it showed a better performance, and they are compared at the last column in appendix II and if the result yields a negative number means that the over all performance of the Trust Region is better, and and visa vers.

5.2 SUMMARY AND CONCLUSION

Optimisation can be classified into two main classes the first called approximation methods, which approximated $f(x)$ to a known function. The second class depends on evaluation of the function f *at* suitable search points. The main difference between the two classes is that approximation methods can only be applied to continuously differentiable functions, while search methods can be applied to any unimodal functions.

Most of the existing Optimisation methods [and indeed methods for solving system of non linear equations] modify and build around Newton's method to create algorithms, which attempt to achieve its rate of convergence.

From the drawbacks of Newton methods the Quasi-Newton method motivated. The basic philosophy of Quasi-Newton method is to have the Newton method rate of convergence without requiring the Hessian be available i.e., its implementation requires only the first derivatives while the second derivative matrix is approximated.

Quasi-Newton methods update at each iteration its Hessian (or inverse) from the current iteration information that will be ignored after reaching the next iteration.

In our implementation we focused on two subiterative Quasi-Newton methods which are Line Search, that generate at each iteration a scalar that regulate the size of search vector to serve as a step from the current iterate to the next one. The second methods which produce a region around x_i that we can trust the minimum to be within. In this method we can directly compute the step vector without doing a focused Line Search on some search direction.

Our tool implement the Trust Region and Line Search methods and it can be plugged in an optimisation packages, and can serves as a comparison benchmark between two classes of optimisation.

Trust Region methods are promising techniques that are proven established conversion theorems

Our numerical results on different problem dimensions revealed that both Trust Region and Line Search produced a comparable performance in small dimensions, but on large dimensions (greater than 80)Trust Region method produced a faster results due to savement in gradient evaluations.

REFERENCE

[1] Currey, H.B. (1944): "The method of Steepest Descent for non linear minimization problems", Quart. Appl. Math., **2** 258-261.

[2] Dennis, Jr. (1979): " Two new unconstrained optimisation algorithms which use function and gradient values".

[3] Fletcher, R. (1987): "Practical Methods of optimisation", vol. 1 of "Unconstraint optimisation", John Wiley and Sons, New York.

[4] Luenberger, D.G. (1984): "Linear And Nonlinear Programming", (second edition), Addison-Wesley, London.

[5] Moghrabi, I.A.R. (1993): "Multi-Step Quasi-Newton methods for optimisation", (Ph.D. Dissertation), Computer Sc. Dept., University of Essex

[6] More, J.J. Garbow, B.S. and Hillstorm, K.E. (1981): "Testing unconstraint optimaisation", A.C.M T.O.M.S., 7 17-41

[7] Naldrett, J.L. (1990): "Trust Region Methods for nonlinear optimisation with function values", (M.Sc. Dissertation), Computer Sc. Dept., University of Essex.

[8] Powell, M.J.D. (1964): "An efficient method for finding the minimum of a function of several variables without calculating derivatives", Computer J. 7, 155-162.

[9] Powell, M.J.D. (1970): "A hybrid method of nonlinear equations", in "Numerical Methods for non linear algebraic equations", (ed. P. Rabinowitz), Gordon and Breach, London.

APPENDIX I

The functions listed here can be found in More et al (1981), unless stated otherwise.

One- The Discrete Boundary value problem:

$$f = \sum_{i=1}^{n} \left[\frac{2x_i - x_{i-1} + x_{i+1} + h^2 (x_i + t_i + 1)^3}{2} \right],$$

Where $h = \frac{1}{n+1}, t_i = ih.$

1) $x_i = t_i * (t_i - 1).$
19) $x^T = ([25,-25]*).$

Two- Broyden Tridiagonal function

$$f = \sum_{i=1}^{n} \left[(3 - 2x_i)x_i - x_{i-1} - 2x_{i-1} + 1 \right]^2.$$

3) $x_i = -1.$

Three- Dixon function (n=10)

$$f = (1 - x_1)^2 + (1 - x_{10})^2 + \sum_{i=1}^{9} (x_i^2 - x_{i+1})^2.$$

4) $x_i = -1.$

Four- The Linear Rank-1 function

$$f = \sum_{i=1}^{n} \left[i \left[\sum_{j=1}^{n} jx_i \right] - 1 \right]^2$$

5) $x_i = 1.$

Five- Full set of Distinct Eigenvalues problem

$$f = (x_i - 1)^2 + \sum_{i=2}^{n} i(2x_i - x_{i-1})^2.$$

2) $x_i = 1.$

Six- Oren and Spedicato Power function.

$$f = \left[\sum_{i=1}^{n} i x_i^2\right]^2$$

6) $x_i = 1.$

Seven- Generalised Powell Quartic function.

$$f = \sum_{i=1}^{\frac{n}{4}} \left[(x_{4i-3} + 10x_{4i-2})^2 + 5(x_{4i-1} - x_{i-1})^2 + (x_{4i-2} - x_{4i-1})^2 + 10(x_{4i-3} - x_{4i})^4\right]$$

where n mod 4 = 0.

7) $x^T = ([3,-1,0,1]^*).$

Eight-The tridiagonal function

$$f = \sum_{i=2}^{n} \left[i(2x_i - x_{i-1})^2\right]$$

8) $x_i = 1.$

Nine- The variably Dimensioned problem

$$f = \sum_{i=1}^{n} (x_i - 1)^2 + \left[\sum_{i=1}^{n} i(x_i - 1)\right]^2 + \left[\sum_{i=1}^{n} i(x_i - 1)\right]^4.$$

9) $x_i = 1 - 1/n.$

10) $x_i = -1.$

Ten- The Generalized Wood Function

$$f=\sum_{i=1}^{\frac{n}{4}}\{100[(x_{4i-2}-x_{4i-3}^2)^2]+(1-x_{4i-3})^2+90[(x_{4i}-x_{4i-1}^2)^2+(1-x_{4i-1})^2]+$$

$$10.1[x_{4i-2}-1)^2+(x_{4i}-1)^2]+19.8(x_{4i-2}-1)(x_{4i}-1)\ \}$$

where n mod 4 = 0

10) x^{T}= ([-300,100]*).

l- Non-diagonal variant of Rosenbrock's function

$$f=\sum_{i=1}^{n}\{\ 100(x_1-x_i^2)^2+(1-x_i)^2\ \}$$

12) x^{T}= ([-1.2,1]*).

m- Penalty function I

$$f=\sum_{i=1}^{n}\{\ 10^{-5}(x_i-1)^2+\left[\left[\sum_{j=1}^{n}x_j^2\right]-1/4\right]^2\ \}.$$

13) x_i=i

n- Modified Trigonometric function

$$f=\sum_{i=1}^{n}\left\{n-\sum_{j=1}^{n}\cos(x_j)+i(1-\cos(x_i))-\sin(x_i)+e^x i-1\ \right\}^2$$

14) x_i=1/n.

o- Penalty II function

$$f=f_1+f_2+f_3$$

where

$$f_1=\sum_{i=1}^{n}(x_i-0.2)^2+\left[a^{1/2}\left[\exp(\frac{x_i}{10})+\exp(\frac{x_{i-1}}{10})-y_i\right]\right]^2,$$

$$f_2 = \sum_{i=n}^{2n} (a^{\frac{1}{2}} \left[\exp(\frac{x_{i-n+1}}{10}) - \exp(\frac{-1}{10}) \right])^2,$$

$$f_3 = \left[\left[\sum_{j=1}^{n} (n-j+1)x_j^2 \right] - 1 \right]^2,$$

where $a=10^{-5}$, y_i =exp(i/20) + exp[(i-1)/10].

15) $x_i = ½$.

P- Generalised shallow function

$$f = \sum_{i=1}^{n/2} \left[(x_{2i-1}^2 - x_{2i})^2 + (1 - x_{2i-1})^2 \right],$$

n mod 2 = 0

16) x_i = -200

s- Extended Rosenbrock's function

$$f = \sum_{i=1}^{n/2} \left[100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \right],$$

n mod 2 = 0.

17) x^T = ([-1.2,1]*).

u- the "Sum of Quartics" problem

$$f = \sum_{i=1}^{25} (x_i - i)^4.$$

18) $x_i = 20$.

APPENDIX II

Please refer to section 5.1 for more details on the notations used in the following tables

	Trust Region					Line Search			Comparison	
Function	NOI	FC		GC		NOI	FC	GC	FC %	GC%
1	13	14		14		9	14	14	0	0
2	10	12		11	*	9	12	12	0	-9.1
3	11	14		12		8	10	10	28.6	16.7
4	18	20		19		17	19	19	5	0
5	2	4		3		2	3	3	25	0
6	30	35	*	31	*	35	41	41	-17	-32
7	30	32		31		28	31	31	3.13	0
8	8	10		9		5	8	8	20	11.1
9	9	11		10		9	10	10	9.09	0
10	14	17		15		7	12	12	29.4	20
12	23	28		24		17	24	24	14.3	0
13	14	18		15		12	14	14	22.2	6.67
14	24	27		25		13	17	17	37	32
15	22	24		23		11	14	14	41.7	39.1
16	203	222	*	204	*	157	241	241	-8.6	-18
17	31	33	*	32	*	32	42	42	-27	-31
18	30	32		31	*	28	32	32	0	-3.2
19	100	102	*	101	*	90	105	105	-2.9	-4
TOTAL	592	655		610		489	649	649	0.92	-6.4

Table I

Results on functions with small dimensions
2<dimensions<16

	Trust Region					Line Search			Comparison	
Function	NOI	FC		GC		NOI	FC	GC	FC %	GC%
1	77	78		78		42	74	74	5.128	5.128
2	49	53	*	50	*	42	69	69	-30.2	-38
3	49	53	*	50	*	42	107	107	-102	-114
4	323	329		324		35	225	225	31.61	30.56
5	50	55		51		6	7	7	87.27	86.27
6	6	8	*	7	*	42	57	57	-613	-714
7	107	128		108		42	73	73	42.97	32.41
8	120	123		121		42	70	70	43.09	42.15
9	49	53		50		23	32	32	39.62	36
10	29	31	*	30	*	42	102	102	-229	-240
12	61	65		62		42	58	58	10.77	6.452
13	82	91	*	83	*	24	98	98	-7.69	-18.1
14	159	164	*	160	*	42	166	166	-1.22	-3.75
15	460	463		461		42	85	85	81.64	81.56
16	174	179	*	175	*	42	390	390	-118	-123
17	5	7	*	6	*	42	82	82	-1071	-1267
18	42	44	*	43	*	42	77	77	-75	-79.1
19	42	44	*	43	*	42	125	125	-184	-191
TOTAL	1884	1968		1902		676	1897	1897	3.608	0.263

Table II

Results on functions with medium dimensions
20<dimensions<50

	Trust Region					Line Search			Comparison	
Function	NOI	FC		GC		NOI	FC	GC	FC %	GC%
1	79	98	*	98	*	67	129	129	-32	-32
2	94	97	*	95	*	86	173	173	-78	-82
3	159	169	*	160	*	65	178	178	-5.3	-11
4	58	63	*	59	*	45	70	70	-11	-19
5	8	10		9		8	9	9	10	0
6	318	348	*	319	*	356	399	399	-15	-25
7	254	258		255		187	242	242	6.2	5.1
8	96	100	*	97	*	85	171	171	-71	-76
9	34	36		35		18	21	21	41.7	40
10	581	605		582		64	173	173	71.4	70.3
12	142	156	*	143	*	117	203	203	-30	-42
13	259	270		260		99	249	249	7.78	4.23
14	673	660		654		162	402	402	39.1	38.5
15	227	232	*	228	*	393	515	515	-122	-126
16	315	323	*	316	*	321	969	969	-200	-207
17	388	393	*	389	*	259	396	396	-0.8	-1.8
18	458	471	*	459	*	375	613	613	-30	-34
19	134	142	*	135	*	105	240	240	-69	-78
TOTAL	4277	4431		4293		2812	5152	5152	-16	-20

Table III

Results on functions with Large dimensions
60<dimensions<85