AMSI
VACATION
RESEARCH
SCHOLARSHIPS
2018-2019

Experiments with Trust Regions and the BFGS Method in Non-smooth Optimisation

Daniel Molent
Supervised by Prof. Andrew Eberhard
RMIT University

Vacation Research Scholarships are funded jointly by the Department of Education and Training and the Australian Mathematical Sciences Institute.

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE
Australian Government
Department of Education and Training

Abstract

This report examines the behaviour of the BFGS method with an inexact line search on nonsmooth functions and its applications to Trust Region philosophy. While many gradient based descent methods are known to perform poorly on non-smooth optimisation problems, the BFGS method has been observed [1, 2, 3] to perform remarkably well and can even converge when tuned appropriately. By examining the path of descent of the BFGS method on the 3-dimensional form of the standard and non-smooth versions of the Rosenbrock function, several inferences are made concerning the behaviour of the algorithm. It was found that the algorithm was quick to locate the valley of the function given by $x_2 = x_1^2$ but took many iterations to locate the global minimum $\mathbf{x} = (1, 1)$. The method demonstrated a strong tendency to follow one side of the valley closely. The region surrounding the valley was associated with poor performance particularly on the non-smooth Rosenbrock function in which the rate of convergence was observed to decrease significantly on approaching the minimum. Modifications were proposed to give the method more freedom and allow it to travel at a distance from the minimum on smoother regions of the function. A non-monotone line search was trialled as an alternative. This allowed the algorithm to enter regions with higher function values further from the valley and select larger step sizes than before. Several trials demonstrated that taking relatively large steps away from the valley of the function were almost always followed by a step towards the previous region indicating the necessity for a descent direction modification. By eigendecomposition of the inverse Hessian approximation D it was observed that the algorithm located and followed the smooth manifold of the function given by the $\mathcal{VU}$ space decomposition closely. A projection of the search direction onto the $\mathcal{V}$ space is proposed and the experiments were mostly successful. The update first assesses whether the $\mathcal{V}$ space has been located by checking whether the associated eigenvalue λ_v is sufficiently small; that is less than a small constant ϵ and projects if the condition is true. The modification accurately detected when to project and the method was found to correct the poor search directions following a step away from the valley on the standard Rosenbrock function. On the non-smooth Rosenbrock the positive-definite property of the D matrix was lost due to the absence of the Wolfe condition and another modification is proposed to use the inexact line search at steps which could intefere with the approximation. A similar approach to rectify issues encountered by the Trust Region on non-smooth functions is proposed to improve the direction in which the quadratic model is fit.

Acknowledgements

I would like to express my gratitude towards my research supervisor Prof. Andrew Eberhard, for the many hours of work he has committed towards guiding the success of this project. I am thankful for his contributions to this report and the use of his own resources, including most of the included algorithms; this report would not have been possible without him. I would also like to thank AMSI and the Department of Education and Training for their work in making this rewarding experience possible and helping undergraduate students to gain a sense of what a career in research may be like.

Introduction

In the field of optimisation, many gradient based descent methods are known to perform poorly on non-smooth functions. Researchers have observed however that the BFGS method demonstrates remarkable performance on non-smooth functions and can even converge if subgradients are used and the line search is tuned appropriately. Although this phenomenon was observed over 25 years ago, little research has been conducted until recently [1, 2, 3] to study the behaviour of the algorithm on non-smooth functions. This report investigates the behaviour of the BFGS method on the non-smooth Rosenbrock function and the standard Rosenbrock function which is used as a point of comparison to identify issues caused by non-smoothness. The eigendecomposition of the inverse Hessian approximation is considered to study the shape of the functions and the manner in which it influences the method's path of descent. To correct the path of descent and prevent the iterates from following regions associated with poor peformance, a modification is proposed to the line search and descent direction. A non-monotone line search is trialled as an alternative to the inexact line search used in the BFGS method to give the method more freedom to travel on smoother regions of the function that previously were not encountered by the iterates except in the first few iterations. Furthermore, $\mathcal{UV}$ space theory is used to modify the poor directions resulting from the non-monotone approach and a similar modification is applied to the Trust Regions. The Trust Region method [5] is an alternative approach which does not use a line search. It instead fits a quadratic model to a function and compares the predicted decrease in function value by the model to the actual decrease as a replacement. Under non-smooth conditions this model encounters issues which are explored in this report and a modification is proposed to rectify the problems.

1 Algorithms

The BFGS and Trust Region algorithms examined in this report are outlined in detail below and their behaviour examined in section 2 and section 4 respectively. As is the case with all optimisation problems, the methods attempt to approximate an input $\mathbf{x}$ to minimise the function $f : \mathbb{R}^n \to \mathbb{R}$ such that:

$$\mathbf{x} \in \arg\min f(\mathbf{x})$$

The BFGS method is a gradient based descent method which uses a direction based on the gradient vector and curvature of the function to descend while Trust Regions select a direction to fit a quadratic model to the function and use a metric to assess its accuracy to locate the minimum. Both algorithms

are iterative and terminate once a stopping criteria is met requiring the gradient of the function to be sufficiently small.

1.1 The BFGS Method

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) method is a popular Quasi-Newton gradient based descent method. The method uses information provided by the gradient vector as well as the local shape ("curvature") of the function to choose a descent direction; it then takes a step of a length determined by a line search and repeats this process until a stopping criteria is met. This technique is loosely analogous to the manner in which a person would locate and walk to the bottom of a hill with their eyes closed. In subsequent iterations, an approximation of the inverse Hessian of the function f is utilised (denoted by D) which becomes more precise at each iteration. This is furthermore used to reduce processing time. The matrix D is used to modify the direction given by $-\nabla f(x^i)$ allowing for a better choice of direction to be made. It takes into account information normally provided by the second order partial derivatives, provided by the Hessian, to account for the local curvature of the function. This is particularly important for the examples considered in this report as steepest descent performs poorly on functions with extreme curvature; such conditions cause the iterates to zigzag. For the directions to remain directions of descent it is critical for the inverse Hessian to remain positive definite. A loss of positive-definiteness will no longer guarantee that a chosen direction will result in a decrease in function value. The Wolfe condition outlined in the following algorithm ensures that this property remains. One of the challenges of this project is how to accommodate this condition and what to do in the event that D looses its positive-definiteness.

The Broyden-Fletcher-Goldfarb-Shanno Method

Initialisation:

1. The function $f : \mathbb{R}^n \to \mathbb{R}$ and its gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ need to be supplied.
2. Initial positive definite symmetric matrix D^0 (usually I and so the first step will be steepest descent) and initial starting point x^0.
3. The tolerance $\varepsilon > 0$, $\sigma \in (0, \frac{1}{2})$ and $\mu \in (\sigma, 1)$.
4. Let $i = 0$.

While: x^i is unsatisfactory i.e. $\left\|\nabla f(x^i)\right\| \geq \varepsilon$ **do**

Step 1: Let $d^i = -D^i \nabla f(x^i)$

Step 2: Solve for t^i for $\min \nabla f(x^i + t^i d^i)$ using the (A) and (W) conditions.

Step 3: Place $x^{i+1} = x^i + t^i d^i$, $s^i = x^{i+1} - x^i = t^i d^i$ and $y^i = \nabla f(x^{i+1}) - \nabla f(x^i)$.

Step 4: Update $D^{i+1} = D^i + \left(\frac{1+\left((r^i)^T y^i\right)}{(y^i \cdot s^i)}\right) s^i (s^i)^T - \left[s^i (r^i)^T + r^i (s^i)^T\right]$ where $r^i = \frac{D^i \mathbf{y}^i}{(y^i \cdot s^i)}$ and increment i by one .

End While

1.1.1 Inexact Line Search

To avoid costly subroutines in minimization in the line search to obtain

$$g(t) := f(x^i + t^i d^i) < f(ix^i) = g(0)$$

for sufficient descent. Instead set our goal to obtain the Amijo condition:

$$\begin{aligned} g(t) = f(x^i + t^i d^i) &< f(x^i) + t^i \sigma \left(\nabla f(x^i) \cdot d^i\right) \\ &= g(0) + t^i \sigma g'(0), \end{aligned} \tag{1}$$

for some $\sigma \in (0, 1)$.

We also demand the Wolfe condition to prevent small steps and to keep the BFGS approximation D positive definite.

$$\nabla f(x^i + t^i d^i) \cdot d^i \geq \mu \nabla f(x^i) \cdot f d^o, \tag{2}$$

holds for the choice of t^k where $\mu \in (0, 1)$. Define, as usual, $g(t) = f(x^i + t^o d^o)$ then (2) is equivalent to

$$g'(t^o) = \nabla f(x^o + t^i d^i) \cdot d^i \geq \mu \nabla f(x^o) \cdot d^o = \mu g'(0). \tag{3}$$

To explain (3) we first observe that $g'(0) = \nabla f(x^i) \cdot d^i < 0$ for d^i to be a descent direction. This implies

$$g'(0) < \mu g'(0) < 0$$

and as $g'(t) \to g'(0)$ as $t \to 0$ we cannot have t too small. Indeed if we assume t is arbitrarily small we will arrive at a contradiction since

$$g'(t) \to g'(0) < \mu g'(0).$$

That is for t small we must have $g'(t) < \mu g'(0)$ contradicting (3).

Line Search Procedure using the Amijo—Wolfe Conditions

Here we outline the pseudocode for the inexact line search.

Inexact Line Search Algorithm: We assume continuous differentiability of g.

1. The initial step length to try $t_0 > 0$.
2. Initially $t_{lo} = 0$ and $t_{hi} = +\infty$ (and so $t_{lo} < t_0 < t_{hi}$).
3. The function $g : \mathbb{R} \to \mathbb{R}$.
4. The two constants σ, $\mu \in (0, 1)$ such that $0 < \sigma \leq \mu < 1$.
5. The maximal number of iterations $K_{\max}$.

Notation: We use $x \leftarrow y$ to mean that the current value of y is to be assigned to the variable x.

Denote the two conditions via:

$$g(t) < g(0) + t\sigma g'(0) \tag{AG}$$

$$g'(t) \geq \mu g'(0) \tag{W}$$

While: (AG) fails or (W) fails and (iteration number $k \leq K_{\max}$) **do:**

If (AG) fails **then**

$t_{hi} \leftarrow t_k$

$t_k \leftarrow \frac{1}{2}(t_{lo} + t_{hi})$

t_{lo} unchanged

else If (W) fails **then**

$t_{lo} \leftarrow t_k$

$$t_k \leftarrow \begin{cases} \frac{1}{2}(t_{lo} + t_{hi}) & \text{if } t_{hi} < +\infty, \\ 2t_k & \text{otherwise.} \end{cases}$$

t_{hi} unchanged

end if

end if

end while

Many gradient methods are of the form

$$x^{i+1} = x^i - t^k D^k \nabla f(x^i)$$

where $d^k = -D^k \nabla f(x^i)$ is chosen with D^k a positive-definite matrix. Then we find that

$$\nabla f(x^i) \cdot d^i = \nabla f(x^i)^T \; d^i = -\nabla f(x^i)^T D^i \nabla f(x^k) < 0$$

due to the fact that $D^k > 0$. That is, d^i is a descent direction, meaning that a small step will lead to a decrease in function value.

Theorem 1 *Suppose H^0 is positive-definite and t^i , for every $0 \leq i \leq j$, is chosen so that $\mathbf{x}^{i+1}$ satisfies*

$$\nabla f(x^{i+1})^T d^i > \nabla f(x^i)^T d^i. \tag{4}$$

Suppose also that we use the BFGS update. Then D^i positive-definite implies D^{i+1} is also positive-definite (and hence all of $d^i = -D^i \nabla f(x^i)$ for $0 \leq i \leq j+1$ are descent directions).

Remark 2 *When the Wolfe condition holds, then*

$$\nabla f(x^{i+1})^T d^j \geq \mu \nabla f(x^i)^T d^i > \nabla f(\mathbf{x}^i)^T d^i$$

since $\mu \in (0,1)$ and $\nabla f(x^i)^T d^i < 0$ (hence (4) holds). For this reason we have tried to enforce the Wolfe condition when negative-definiteness of the BFGS approximation becomes an issue. When one does not explicitly enforce this the D^i can have negative eigenvalues which effects convergence, and descent properties of the algorithm.

1.2 Trust Region

The Trust Region method adopts an entirely different approach to minimising functions than gradient based techniques, like the BFGS method. Instead of choosing a direction to step in, a direction is chosen to fit a quadratic model to the function which approximates its shape locally. This is chosen to be the Taylor expansion around a point x^k up to the second order:

$$m_k(d) := f\left(x^k\right) + \left(g^k\right)^T d + \frac{1}{2} d^T B^k d$$

If the predicted model decrease $m_k(0) - m_k(d^k)$ is good compared to the actual decrease in function value $f(x^k) - f(x^k + d^k)$ the model prediction can be trusted within the bounds of the trust region and it is safe to take a step and even increase the size of the trust region if the model is highly

accurate. If the model prediction is not accurate the size of the trust region must be reduced and a step is not taken. The trust region can be visualised as a spherical region surrounding an iterate in which the accuracy of the model can be trusted. This process removes the need for a line search and replaces it as an alternative. This technique is known to be a powerful, all-round method suitable for use on a variety of functions with varying conditions. It does not necessarily have to be accurate in its descent to produce high quality and reliable results which makes it a suitable choice for the functions considered in this report. An approach to choosing the direction known as the "Dog-Leg" (see Appendix A) used in this report also utilises information provided by the BFGS Hessian approximation to account for the local shape of the function when fitting the quadratic model.

The Trust Region Method

We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is partially smooth. Let $g^k \in \partial f\left(x^k\right)$ at each iteration (often this will mean $g^k = \nabla f\left(x^k\right)$).

Initialization:

The model function m_k needs to be supplied, along with initial $\bar{\Delta}$, $\gamma > 0$, $\Delta_0 \in (0, \bar{\Delta})$ and $\eta \in [0, \frac{1}{4})$; initially $D^0 = I$ and $d^0 = -g^0$.

While: x^k is unsatisfactory (i.e. $\left\|d^k\right\| \geq \varepsilon$?) **do**

Step 1: Obtain d^k by *approximately* solve the problem

$$\min_d m_k(d) := f\left(x^k\right) + \left(g^k\right)^T d + \frac{1}{2} d^T B^k d$$
$$\text{Subject to } \|d\| \leq \Delta^k. \qquad \text{(OPT:Trust)}$$

Step 2: Calculate $\rho_k := \frac{f\left(x^k\right) - f\left(x^k + d^k\right)}{m_k(0) - m_k\left(d^k\right)}$.

Step 3: **If:** $\rho_k < \frac{1}{4}$

$\Delta^{k+1} = \frac{1}{4}\left\|d^k\right\|$

else

If $\rho_k > \frac{3}{4}$ and $\left\|d^k\right\| = \Delta^k$ (*)

$\Delta^{k+1} = \min\left(2\Delta^k, \bar{\Delta}\right)$

else

$\Delta^{k+1} = \Delta^k$

end

end

If $\rho^k > \eta$

$x^{k+1} = x^k + d^k$ and update approximation B^{k+1}.

else

$x^{k+1} = x^k$

end

End While

In the trust region algorithm we set a size Δ within which we want to *trust* the model to predict a descent by solving (OPT:Trust). If the model fits well on the trust region, then the predictions will be good and this is measured by the quantity ρ^k the ratio of the actual to predicted descent (note that the denominator is always negative as d^k solves (OPT:Trust). If ρ^k is small, then we reduce the trust region and update the model to hopefully get a better fit next time around. If ρ^k is very large (close to 1) then we increase the trust region when we have the d^k already hitting the boundary of the trust region.
If ρ^k is greater than a thresh hold η we make the update and recalculate all relevant quantities. Otherwise we reject the step and work on improving the model.

2 Behaviour of the BFGS Method

To understand the behaviour of the BFGS method the path of descent is first considered. To explore the path of the iterates 2-dimensional contour plots were generated to examine the minimisation of 3-dimensional functions. The standard Rosenbrock function and the non-smooth Rosenbrock function are considered in this report; the standard version is a point of reference used to identify issues associated with the performance on non-smooth functions. The Rosenbrock function is a function commonly used to test the performance of Optimisation methods. The valley of the function given by $x_2 = x_1^2$ is easy to locate; however the minimum is rather difficult to find in comparison. In the following figures the valley is marked in red, the iterates are marked by white x's and their paths of descent joined with purple lines.

2.1 Rosenbrock Function

The Rosenbrock function is a smooth function with a banana shaped valley in which the minimum is located. The general equation $f : \mathbb{R}^n \to \mathbb{R}$ is given by:

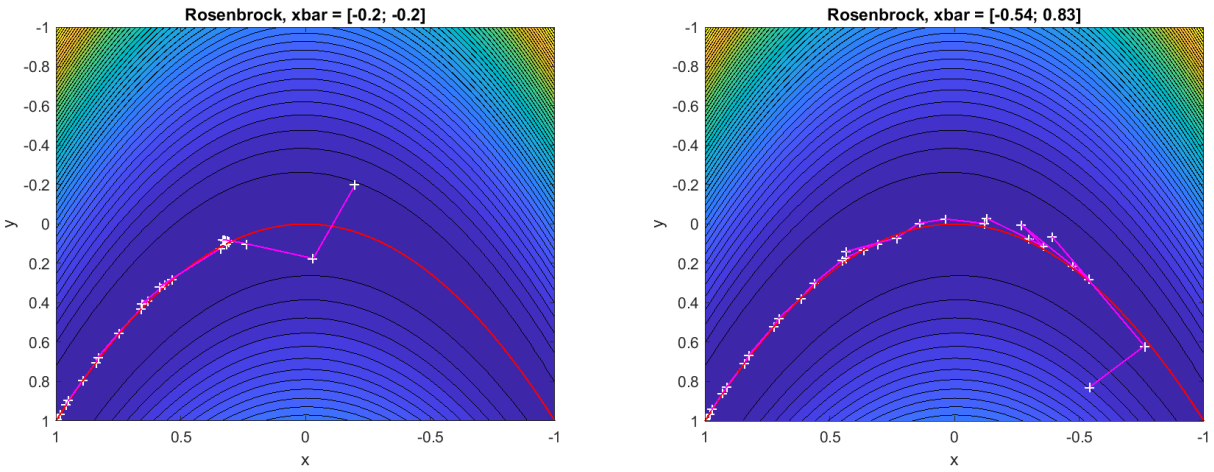
$$f(\mathbf{x}) = \sum_{i=1}^{N-1} (1 - x_i)^2 + w(x_{i+1} - x_i^2)^2$$

where $w \in \mathbb{R} : w > 0$. To explore the behaviour of the BFGS method on this function by the use of visualisations, the 3-dimensional form of this function is examined given by:

$$f(\mathbf{x}) = (1 - x_1)^2 + w(x_2 - x_1^2)^2 \quad \{w \in \mathbb{R} : w > 0\}$$

The constant $w = 100$ is chosen in the examples following and the minimum of the function is situated at $(1, 1)$.

In the sample of random starting points examined the overall performance was generally quite good, especially considering the function is known to be quite difficult to minimise. The method converged to the correct solution of $(1, 1)$ in all instances using $\sigma = 0.1$ and $\mu = 0.9$ in the inexact line search. Within a few iterations at most, the algorithm was able to successfully locate the valley. The search

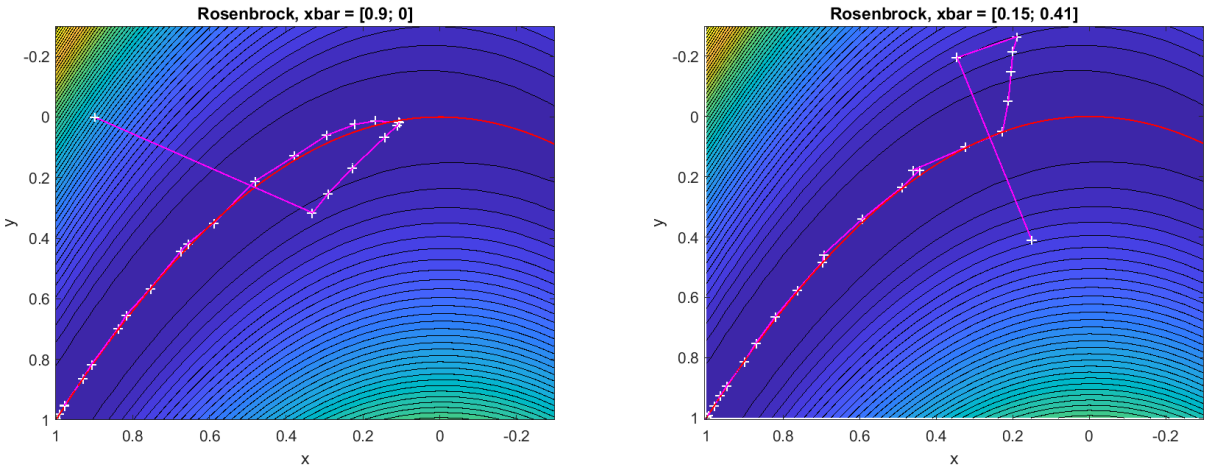


(a) Rosenbrock 1

(b) Rosenbrock 2

Figure 1

directions, while they underwent a brief period of initial instability during the first several iterations were stable and the iterates demonstrated a tendancy to follow the valley in a tangential manner, particularly on the side given by $\{x \mid x_2 < x_1^2\}$. Regions of the functions other than the valley were seldomly encountered as the step sizes were relatively small, usually in the direction along the valley.



(a) Rosenbrock 3

(b) Rosenbrock 4

Figure 2

The step lengths were reasonably consistent in size. In certain instances the descent directions were abstract and the iterates went away from the minimum in some instances; however these steps were generally smaller in comparison and generally did not result in a significant loss in progress. In some circumstances these directions resulted in a minor loss of progress from back-tracking, or became stuck in a region of the function for a brief period. In one trial the iterates stepped away from the minimum for several iterations before turning in a V-like direction to follow the valley towards the minimum. While this kind of behaviour may seem impossible at first given that the function values must always decrease, the iterates may travel away from the minimum in the x_1 direction by stepping closer to the valley at each iteration; this ensures that the Armijo condition is satisfied as the function value still decreases sufficiently. It is unclear why such search directions were chosen in this example.

2.2 Non-smooth Rosenbrock Function

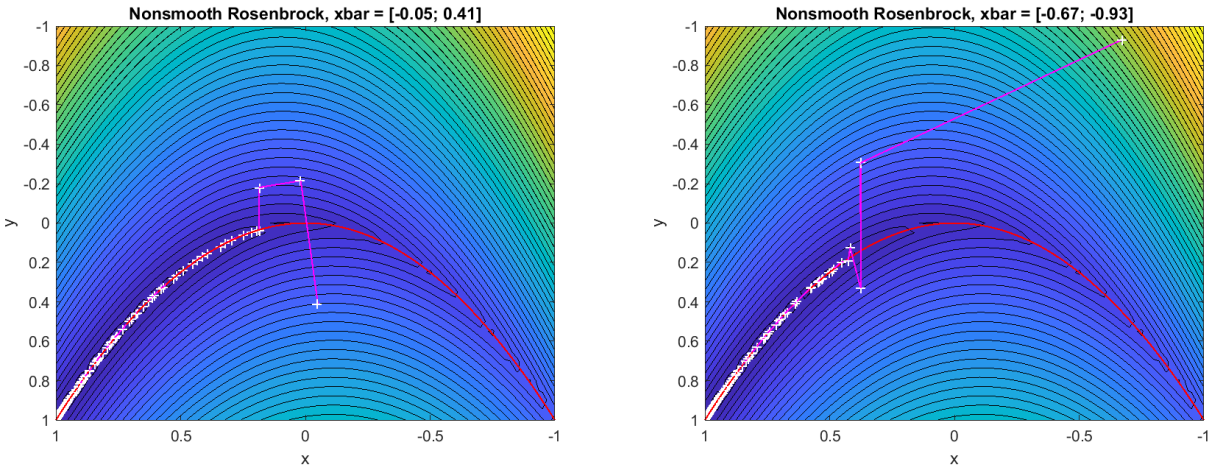
The non-smooth Rosenbrock is a non-smooth variant of the Rosenbrock function. The general equation $f : \mathbb{R}^n \to \mathbb{R}$ is given by:

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} (1 - x_i)^2 + w|x_{i+1} - x_i^2|$$

where $w \in \mathbb{R} : w > 0$. Similar to the standard Rosenbrock function examined in section 2.1, the 3-dimensional form of the function is considered given by:

$$f(\mathbf{x}) = (1 - x_1)^2 + w|x_2 - x_1^2| \quad \{w \in \mathbb{R} : w > 0\}$$

The constant $w = 10$ is chosen as it scales the figures closely to those in section 2.1. The minimum of the function is similarly situated at the point $(1, 1)$. This function differs from the standard Rosenbrock function in that the valley of the function $x_2 = x_1^2$ is non-differentiable, the sides of the valley are extremely steep and it is subsequently more difficult for gradient based methods to minimise it. Since the descent method relies on the gradient information at each iteration, an error will be encountered in the event that an iterate lands directly on the valley. While this initially seems problematic, the likelihood of an iterate landing on a non-differentiable point is highly unlikely using an inexact line search and for this reason the situation is not accounted for in the method. An exact line search is not considered in this report for similar reasons; any attempt to minimise this function with such a line search will result in the method locating a non-differentiable point within the first several iterations and the method subsequently failing.



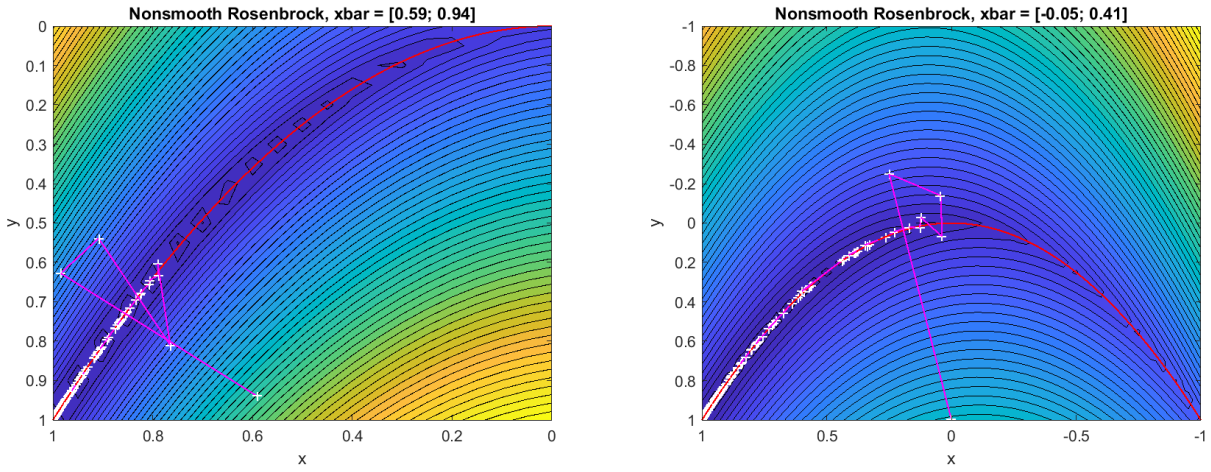
(a) Non-smooth Rosenbrock 1

(b) Non-smooth Rosenbrock 2

Figure 3

The algorithm encountered significantly more problems minimising this function compared to the standard Rosenbrock function. The sample of random starting points trialled demonstrated similar properties to the Rosenbrock function; however the issues were exacerbated on the non-smooth version. In all instances trialled the method converged to the correct solution of $(1, 1)$; however the rate of convergence slowed down considerably upon closely approaching the minimum and took multiple times the number of iterations to converge. The descent directions within the first several iterations similarly demonstrated instability initially; and then settled down, following the valley tangentially mostly on the side given by $\{x \mid x_2 < x_1^2\}$. The iterates were observed to back-track both regularly and periodically along the valley away from the minimum for a few iterations. This

was partially responsible for the extremely slow rate of convergence. It is not clear why such directions were chosen although it is likely to be a result of the surface in this region closely surrounding the valley being less smooth and more extreme than the smooth version of the function. It may alternatively be due to a failure to retain positive definiteness of D^i and an associated failure to determine a descent direction. While it may appear that a trajectory leading away from the minimum should not be possible given that the Armijo-Goldstein condition enforces a strict decrease in function value, the condition can be satisfied by stepping sufficiently closer to the valley at each iteration as discussed in section 2.1. This behaviour may furthermore be partially responsible for the



(a) Non-smooth Rosenbrock 3

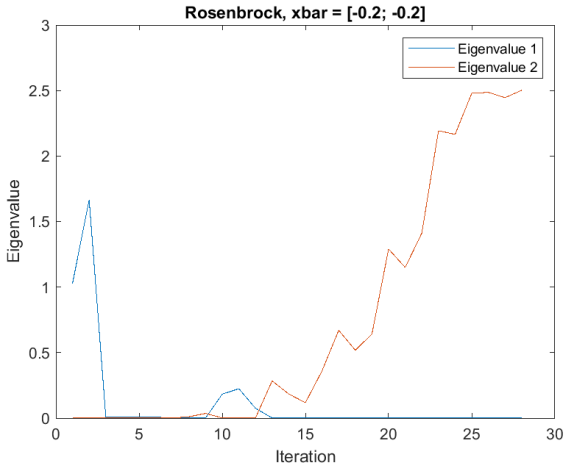
(b) Non-smooth Rosenbrock 4

Figure 4

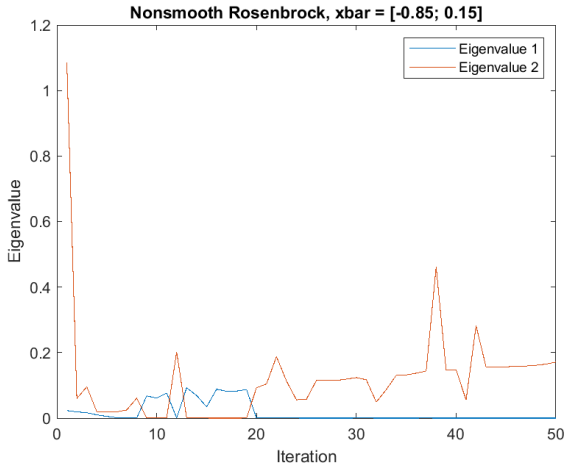
slow rate of convergence. For a point x_i, the set of potential next points $\{x^{i+1} := x_i + d^i | f(x^i + d^i) \leq f(x^i) + \sigma \nabla f(x^i) \cdot d^i\}$ (ignoring the Wolfe condition) decreases as i increases. As the iterates progress, increasingly larger regions of the function become inaccessible as the function values in these regions are too large to ever satisfy the Armijo condition. While this did not noticeably impact the performance on the Rosenbrock function, it is possible that the delays caused by the iterates back-tracking in the non-smooth version is causing the inaccessible region to grow too quickly. This restricts the iterates to the immediate region surrounding the valley and explains why the stepsize is restricted significantly; as the step sizes increase they will be quickly rejected as the Armijo condition will be violated meaning that only tiny step sizes are viable.

2.3 Eigendecomposition of BFGS Inverse Hessian Approximation

To further understand the behaviour of the BFGS method on the functions considered in section 2 the search directions were examined further. To gain insight into the choice of direction, the inverse Hessian approximation D was examined by eigendecomposition; that is by finding the set of scalars λ and non-zero vectors $\mathbf{v}$ such that: $D\mathbf{v} = \lambda\mathbf{v}$. By examining the eigenvalues and eigenvectors of the function a deeper understanding may be gained into both the structure of the functions and the modification of the search directions. For several random starting points, the matrix was decomposed into eigenvalues and eigenvectors at each iteration. The results were consistent with the



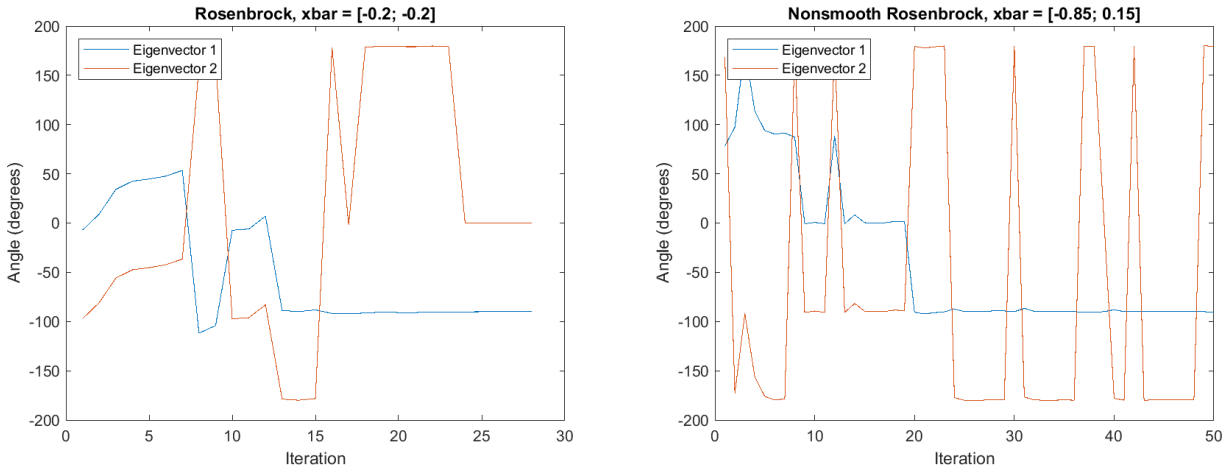
(a) Rosenbrock: Size of eigenvalues of D^i.



(b) Non-smooth Rosenbrock: Size of eigenvalues of D^i.

Figure 5

points trialled. The first eigenvalue λ_1 (coloured blue) rapidly decreased within the first few iterations and approached zero. The values were observed to spike randomly, particularly within the first 15 - 20 iterates, but still demonstrated a downwards trend overall. The random spikes were more frequent on the non-smooth Rosenbrock function. The second eigenvalue λ_2 (coloured orange) was found to also decline in value initially; however it then increased in size as the minimum of the function was approached. To understand the significance of these results the direction of the eigenvectors must also be considered. The tangent to the valley $x_2 = x_1^2$ is used as a point of reference and the directions expressed as the angle formed with the tangent vector with postive x_1 component. The eigenvectors were found to always be orthogonal. This result was expected as the inverse Hessian is a symmetric matrix, and symmetric matrices are known to possess orthogonal eigenvectors. The eigenvectors went through an inital stage of instability, similar to the behaviour



(a) Rosenbrock: Angles eigenvectors make to the tangent.

(b) Non-smooth Rosenbrock: Angles eigenvectors make to the tangent.

Figure 6

observed in section 2; after stabilising, the two eigenvectors aligned roughy parallel and perpendicular to the tangent to the valley. The relatively small eigenvalue λ_1 corresponds to the eigenvector in the direction perpendicular to the tangent $\mathbf{v_1}$ and the eigenvalue λ_2 corresponds to the direction parallel to the tangent $\mathbf{v_2}$. To consider what these results imply about the structure of the functions the factorisation $D = Q^T \Lambda Q$ is considered where Q is a matrix with eigenvectors $[v_1, v_2, ..., v_n]$ and Λ is an $n \times n$ square matrix with diagonals $[\lambda_1, \lambda_2, ..., \lambda_n]$. The inverse of D, i.e. the Hessian of f is given by $D^{-1} = Q\Lambda^{-1}Q^T$. To calculate the inverse of the eigenvalue matrix Λ^{-1} the reciprocal of each eigenvalue is taken.

$$\begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \ddots & \dots & 0 \\ 0 & \dots & \dots & \lambda_n \end{bmatrix}^{-1} = \begin{bmatrix} \lambda_1^{-1} & 0 & \dots & 0 \\ 0 & \ddots & \dots & 0 \\ 0 & \dots & \dots & \lambda_n^{-1} \end{bmatrix}$$

This approximation of the eigenvalues associated with the Hessian indicate that the second order rate of change in the $\mathbf{v_1}$ direction is effectively infinite. As λ_1^{-1} approaches zero , λ_1 approaches ∞ suggesting that the gradient of the function changes instantly in the $\mathbf{v_1}$ direction. This is indicative of non-smoothness. The second order rate of change is comparatively normal in the $\mathbf{v_2}$ direction indicating smoothness in this direction. These results concur with recent $\mathcal{VU}$ space theory which suggests that non-smooth functions are partially smooth in that there exists a smooth manifold and perpendicular to this smooth section exists a non-smooth, V-shaped surface. Locally around a point in the smooth manifold the space can be decomposed into the non-smooth $\mathcal{V}$-space and the smooth

$\mathcal{U}$-space (defined by the smooth section). The curvature across the surface is infinite in the V-direction and finite in the U-direction. It was observed in section 2.1 that the iterates tended to follow the valley in an approximately tangential direction. These findings furthermore suggest that the algorithm is attempting to locate the smooth manifold of the function during the observed stages of initial eigenvector instability and follow it towards the minimum.

3 BFGS Method Updates

By examining the behaviour of the BFGS method in section 2 it was apparent that performance issues were encountered by following the valley of the functions closely. This is particularly true for the non-smooth Rosenbrock function as the rate of convergence was observed to decrease significantly on approaching the minimum. To rectify this issue and improve the performance of the algorithm, a feasible approach may be to modify the algorithm to avoid following the non-smooth region in and surrounding the valley so closely. By following the valley at a distance on the smoother regions of the function where the performance is known to be better, the issues encountered may be mitigated in theory. The Armijo-Goldstein condition used in the inexact line search was believed to be primarily responsible for the methods tendency to follow the valley; however the search directions were also found to be partly responsible. To create a viable modification to the BFGS method both the line search and choice of direction must be considered. The modifications have been developed and fine-tuned on the standard Rosenbrock function first before testing it on the non-smooth Rosenbrock.

3.1 Non-monotone Line Search

From exploring the behaviour of the descent method in section 2 it was found that the smoother regions of both Rosenbrock functions were seldom encountered by the algorithm; it instead favoured the valley of the functions and followed them closely towards the minimum. The Armijo-Goldstein condition used in the inexact line search enforces a strict decrease in function value at each iteration, limiting the set of viable next iterates x_{i+1} to increasingly smaller regions surrounding the valley. It is not possible for the algorithm to take a large step and place the next iterate out of the valley. To allow the iterates to travel on the higher regions of the function the Armijo condition must be weakened to enforce a less strict decrease in function value. An alternative line search technique known as a non-monotone line search [4] was trialled for this purpose.

Non-monotone Line Search

The function g is assumed to be continuously differentiable.

Initialisation:

1. A point x_i, search direction d_i, function value f_i and gradient g_i must be supplied; and
2. We set an integer $M \geq 0$ of previous function values to consider, $\sigma \in (0,1)$, $\gamma \in (0,1)$ and $m(0) = 0$.

Step 1: Set $t = 1$

Step 2: Compute $f_t = f(x_i + td_i)$. If

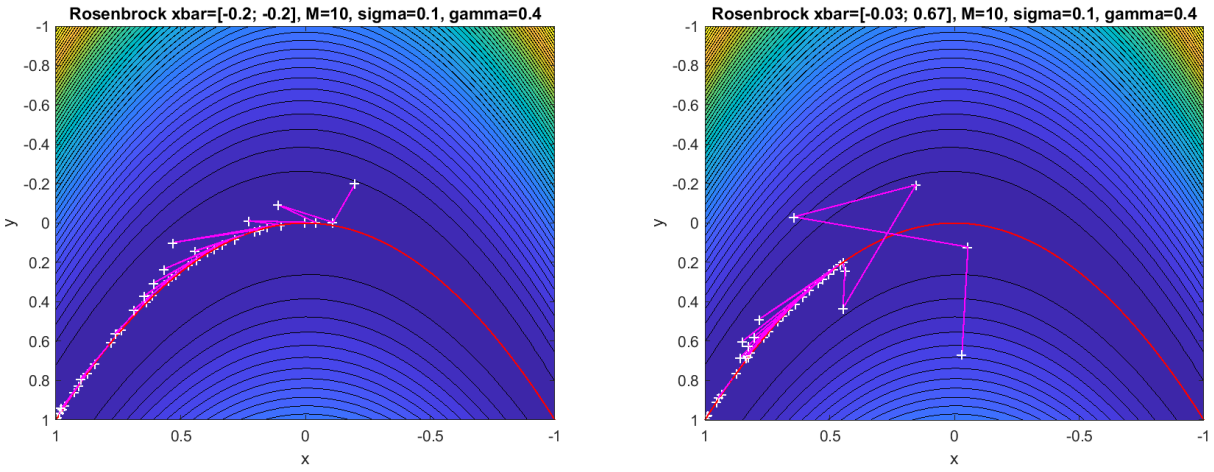
$$f_t \leq \max_{0 \leq j \leq m(i)} f_{i-j} + \sigma t g_i^T d_i$$

set $f_{i+1} = f_t, x_{i+1} = x_i + td_i, i = i + 1, m(i) = \min[m(i-1), M]$ and **return**.

Step 3: Set $t = \gamma t$ and go to Step 2.

Typical parameters for this technique are $\sigma = 0.1$, $\gamma = 0.4$ and $M = 10$.

A non-monotone approach to choosing the step length t possesses two key advantages over the former inexact line search used. Instead of considering the previous function value f_i the maximum within the set of the M previous function values is considered. This weakened condition means that new function values do not necessarily have to be an improvement from the previous function value; as long as it is no worse than the maximum of the Mth previous iterates (plus the multiple of the directional derivative) then it is a viable point. This gives the method more freedom to travel further away from the valley and even increase in function value from the previous iteration while still preserving the property of convergence. The technique also favours larger step sizes decreasing the likelihood of iterates being close to the valley. This is achieved by starting with a relatively large step size $t = 1$ and gradually reducing the size of t by a constant $\gamma \in (0,1)$ until the line search conditions are satisfied. In the random set of starting points trialled the non-monotone line search enabled the iterates to step away from the valley as predicted. Over time, the distance the points strayed from the valley gradually decreased and the algorithm converged to the correct solution of $(1,1)$. The factor γ was highly sensitive to change and was difficult to tune; larger values caused the iterates to jump around erratically while smaller values resulted in tiny steps and minimal progress. Tuning the parameter perfectly resulted in the algorithm first entering the valley, then taking a large step away from it in the direction approximately tangential to the valley and finally taking a similarly sized step



(a) Non-monotone 1

(b) Non-monotone 2

Figure 7

towards the valley placing the iterate into a position close to the last iterate positioned in the valley. This behaviour repeated continually until the algorithm converged. While creating a modification that enabled the iterates to leave the valley was a success, any progress made was immediately ruined by the next choice of direction as it brought the iterates away from the minimum.

3.2 Projection of the Descent Direction

To produce a viable algorithm the descent direction must be considered and appropriate modifications made. The replacement of the inexact line search with a non-monotone line search explored in section 3.1 resulted in some search directions which placed iterates farther from the minimum and closer to the valley. To rectify the issue, the information from the eigendecomposition of the inverse Hessian approximated in section 2.3 can be used. From the theory of the $\mathcal{VU}$d ecomposition we know that for *partially smooth* functions [9, 8] one can locally decompose the domain into two orthoganal subspaces. Within one subspace the function looks locally smooth, while in the other the function have a *strong local minimum* of a strong stable category. This is the so called $\mathcal{V}$ space where the graph has this indicative shape. As we have noted in section 2.3 these subspaces may be inferred from the eigenvector decomposition of the inverse Hessian approximation provided by the BFGS approximation D. The eigenvectors associated with the near zero eigenvalues of D correspond to the directions in which the Hessian (approximated by D^{-1}) has a near infinite curvature. These are in the $\mathcal{V}$ space. The eigenvectors associated with positive and bounded eigenvalues span the $\mathcal{U}$ space.

After locating the $\mathcal{VU}$ space of the function, the eigenvectors associated with the D matrix align with the directions parallel and perpendicular to the tangent to the valley $x_2 = x_1^2$ as discussed in section 2.3. Instead of accepting a poor choice of direction, the descent direction can instead be projected onto the $\mathcal{V}$ space resulting in an updated search direction that is approximately perpendicular to the valley. Although it may seem counter-intuitive to project towards the valley, the behaviour of the algorithm will ensure that the next search direction is approximately tangential, aligning with the $\mathcal{U}$ space and allowing the function to take a large step into the smoother regions. For this approach to work it is necessary to recognise whether the $\mathcal{VU}$ space has been located; projecting before this will result in the projection not aligning with the $\mathcal{V}$ space and will likely be a poor choice of direction. The small eigenvalue λ_v is indicative of whether or not the $\mathcal{V}$ space has been located; values sufficiently close to zero (which is defined later as being less than a small constant $\epsilon > 0$) indicate an infinite second order rate of change in the direction of the associated eigenvector $\mathbf{v}$ and hence non-smoothness.

Projection along the V-space given by the BFGS Method

Function $[z, U, \lambda, V, \mu] = P_{\mathcal{V}}(x^0, D)$ We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is twice continuously differentiable and a small constant $\epsilon > 0$.

Initialisation:

1. The positive definite symmetric matrix D needs to be supplied; and
2. an initial starting point x^0.

Step 1: Calculate the eigenvectors and values of D i.e. $Dx_i = \lambda_i x_i$ (where $\lambda_i \geq 0$)

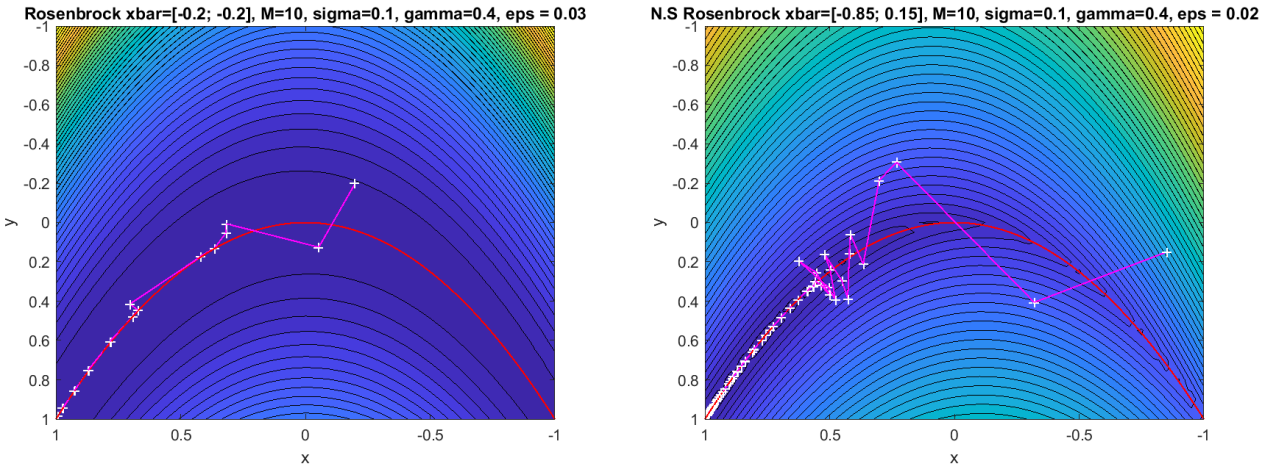
Step 2: Partition the eigenvectors into two groups based on the size of λ_i. Let $i = 0$

1. (a) For all k while $\lambda_k \leq \epsilon$ place $v_{i+1} := x_k$ and $\mu_{i+1} := \lambda_k$ then $V := \{v_1, v_2, \ldots, v_m\}$
   (b) Then $U := \{x_1, x_2, \ldots, x_n\} \cap V^c$ and λ the associated multiplier .
   (c) This results in $\mathcal{V} = \text{span}\{v_1, v_2, \ldots, v_m\}$ and $\mathcal{U} = \text{span}\{u_1, u_2, \ldots, u_{n-m}\}$

Step 3: Projection along $\mathcal{V}$ space: Let $\hat{v}_i := \frac{v_i}{\|v_i\|}$ and

$$z = P_{\mathcal{V}} x^0 := \sum_{i=1}^{m} (\hat{v}_i \cdot x_i)\, \hat{v}_i.$$

In section 2.3 it was observed that λ_v became progressively smaller as the iterates approached the minimum. For this reason there is a point where if convergence has not occured all iterates meet the projection critieria. At this stage the algorithm will constantly project d along the $\mathcal{V}$ space, no further progress will be made and the optimal solution will never be found. To correct this behaviour a condition requiring a given point x_i to satisfy $i \in (2k+1)$ where $k \in \mathbb{Z}$ was introduced allowing for search directions to alternate between projections onto the $\mathcal{V}$ space and those given by the standard BFGS method. The update worked particularly well on the standard Rosenbrock function. In most



(a) Rosenbrock Projection

(b) Non-smooth Rosenbrock Projection

Figure 8

instances the algorithm correctly recognised when to project onto the $\mathcal{V}$ space and experiments confirmed that the approach did work in practice. The method first located the valley and then took a relatively large step tangential to the valley onto the smoother regions of the surface. The projections usually occurred immediately after the iterates stepped away from the valley and were in the direction roughly perpendicular to the tangent of $x_2 = x_1^2$ as expected. The method alternated between standard BFGS direction (usually roughly in the tangent direction to the valley) and projections as predicted. Choosing too small a value of ϵ resulted in the non-monotone line search behaviour from section 3.1 initially until λ_v was sufficiently small; values too large resulted in projections occuring before the $\mathcal{V}$ space had been located. In some instances, the projections were able to correct abstract behaviour. In section 2, the starting point $[0.9, 0]^T$ was found to back-track away from the minimum for several iterations (see Figure 2a). The modified BFGS method resulted in a shorter first step and a more direct approach to the minimum. The iterates no longer back-tracked and the solution was located in significantly less iterations. These tremendous

improvements were found to be situational and dependent on the starting point; in general the performance was on par with the standard BFGS algorithm. The method did not function as intended on the non-smooth Rosenbrock function. The iterates were found to zigzag across the valley and continue past the optimal solution onto higher regions of the function. The BFGS method by its design does not produce direction of ascent, indicating that the modifications introduced an error to the algorithm. The inverse Hessian approximation was deemed to be the cause of the issue. To ensure that the chosen direction is a descent direction, the inverse Hessian must be a positive-definite matrix. If this property is lost there is no guarantee that a chosen direction will result in a decrease in function value. By examining the eigenvalues of the D matrix it was found that it no longer possessed positive-definiteness as some of the eigenvalues were negative. The Wolfe condition used in the inexact line search is responsible for retaining positive-definiteness and the absence of this condition in the non-monotone line search was deemed to be responsible for the issue. As an experiment, the line search was replaced with an inexact line search in the instances where projections occurred. It was clear that the projections possessed a destructive influence on the D approximation. A more stable line search with the Wolfe condition would in theory ensure that the matrix remains positive-definite or would at least mitigate the observed effects. The update resulted in a significant improvement in performance; however some of the issues with positive-definiteness remained. In its more stable form the method still demonstrated further issues. The iterates slowed down significantly on close approach to the minimum as they were observed to do in section 2.2. While the non-monotone line search certainly gave the method more freedom it was far from enough. As an experiment to rectify the issue the number of elements in the set containing the previous M function values was doubled every 20 iterations. This modification effectively allowed for larger previous function values to be included in the evaluation of the maximum function value used in the non-monotone line search giving the method the freedom to take larger steps and converge to the solution faster. This update was found to significantly improve the rate of convergence.

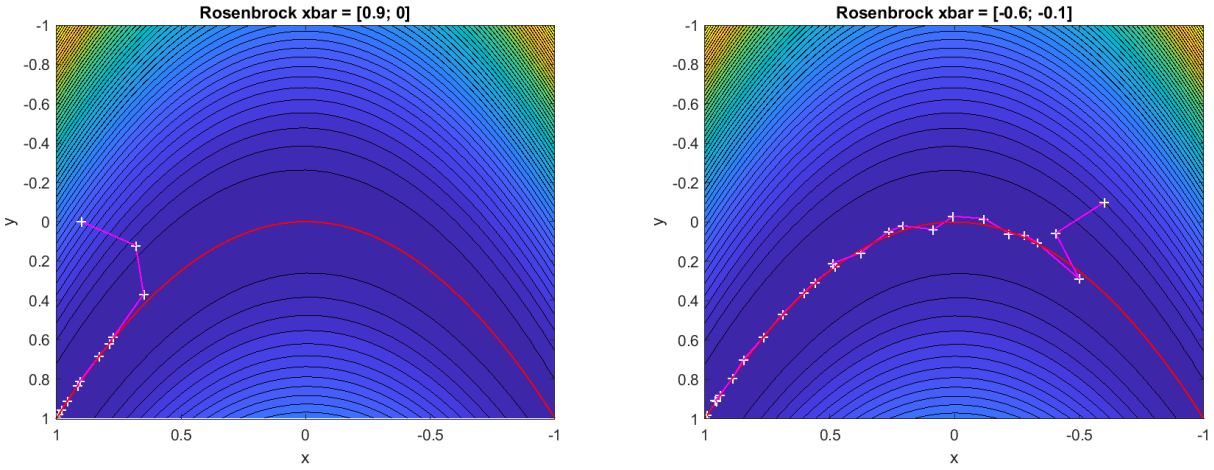
4 Behaviour of the Trust Region method

The Trust Region method's path of descent is similarly explored through use of 2-dimensonal contour plots. The Rosenbrock function and the non-smooth Rosenbrock function examined in section 2.1 and section 2.2 are similarly considered. The performance of the algorithm on the standard and non-smooth versions of the function is compared as well as with the BFGS method examined in section 2. The constants $w = 100$ and $w = 10$ are again used in the equations for the

3-dimensional forms of the Rosenbrock and non-smooth Rosenbrock functions respectively.

4.1 Rosenbrock Function

The standard Trust Region method using the "Dog-Leg" strategy demonstrated good performance on the smooth Rosenbrock function. It took smaller steps in comparison which prevented the



(a) Rosenbrock Trust Region 1

(b) Rosenbrock Trust Region 2

Figure 9

unstable and erratic initial directions observed in the BFGS behaviour and crossed over the valley less. The method demonstrated the same tendancy to follow the valley closely although it did not approach it as aggressively. In some instances the method actually "curved" towards it rather than approaching it directly. In all trialled instances the method located the minimum relatively easily. Overall, the performance of this method was quite good.

4.2 Non-smooth Rosenbrock Function

The performance of the Trust Region method on the non-smooth Rosenbrock function was significantly worse. The method failed to converge to the correct solution of $(1, 1)$ in all instances. The method was found to approach the valley $x_2 = x_1^2$ directly and it stopped once it reached the bottom. It made little to no progress travelling along the valley to the optimal solution. It was rather strange that a method which performed so well on one function could encounter a multitude of performance issues on another similar function. The reason behind the poor performance appearered to be associated with the shape of the valley. The Trust Region method attempts to fit a quadratic model to the function given by the second order Taylor expansion around a point. For iterates on the

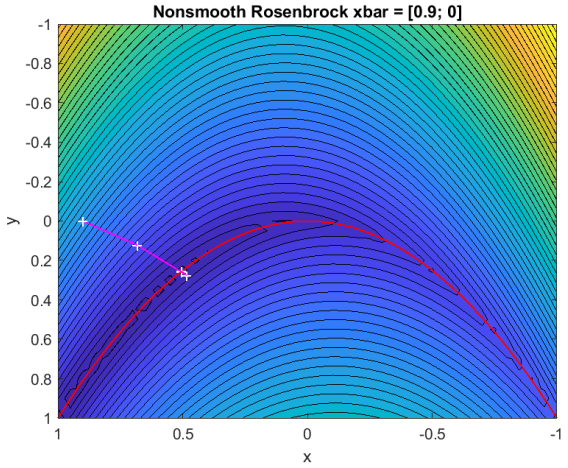


Figure 10: Non-smooth Rosenbrock Trust Region

valley, the quadratic model fitted to the function lies along the valley in an approximately tangential direction. The valley of the function however is extremely narrow causing any inaccuracies in the chosen direction to result in a quadratic model which almost immediately collides with the sides of the valley. This results in the steps taken by the method being extremely small as seen in the plot. A simple analogy is to consider how a person would navigate through an extremely narrow corridor in the dark. The person would first choose a direction with some uncertainty. If the direction is not precise the person will immediately hit the side of the corridor. For the method to work well the directions must be perfect; this is however rarely the case in practice as observed in this experiment.

5 Modification of Trust Region Method

The narrowness of the valley was found to stop the convergence of the Trust Region method on the non-smooth Rosenbrock in section 4.2. By using an approach similar to the BFGS $\mathcal{V}$ space projection in section 3.2 the error in the model direction may be rectified in theory. Since the direction of extreme curvature in the valley is in the direction of the $\mathcal{V}$ space, this direction can be identified and used to modify the direction of the quadratic model to prevent it from colliding with the sides.

5.1 Modified Trust Region using $\mathcal{VU}$ decomposition

In non-smooth optimisation the gradient is often replaced with the subdifferential. The theory of subdifferentials is extensive and involved so for our purposes we will assume f is Lipschitz continuous and that the function is sufficient to consider the differential to coincide with the

so-called Clarke subdifferential.

$$\partial f(x) := \text{comv}\{g \mid g = \lim_{x' \to x} \nabla f(x') \text{ for } x' \in D(f)\},$$

where $S(f)$ denotes the set of full measure on which the gradient exists and comv denotes the convex hull of the set (the smallest closed convex set containing the given set). When f is partially smooth and a $\mathcal{VU}$ is available around x then it may be shown that

$$\partial f(x) = \{g_{\mathcal{U}}\} \times \partial_{\mathcal{V}} f(x)$$

where $g_{\mathcal{U}} = P_{\mathcal{U}} g$ for any $g \in \partial f(x)$ and $\partial_{\mathcal{V}} f(x) = \{P_{\mathcal{V}} g \mid g \in \partial f(x)\}$. We assume that we may sample $g \in \partial f(x)$ for any given point x (usually this will just be a gradient of an "active component" of f i.e. the limit of a nearby gradient). We note that its is true that the one side directional derivative

$$f'(x, d) := \liminf_{\substack{t \downarrow 0 \\ d' \to d}} \frac{1}{t}(f(x + td') - f(x)) = \max\{d^T d \mid g \in \partial f(x)\}.$$

A necessary condition for a local minimum is $0 \in \partial f(x)$ or $f'(x, d) \geq 0$ for all d. In our case we will use the information provided by the $\mathcal{VU}$ decomposition. As f is essentially smooth in the $\mathcal{U}$ direction, we may project the gradient onto this direction and considert if this component $g_{\mathcal{U}} = 0$.

The Basic Non-smooth Trust Region Method

We assume that $f : \mathbb{R}^n \to \mathbb{R}$ is partially smooth. Let $g^k \in \partial f\left(x^k\right)$ at each iteration (often this will mean $g^k = \nabla f\left(x^k\right)$).

Initialisation:

The model function m_k needs to be supplied, along with initial $\bar{\Delta}$, $\gamma > 0$, $\Delta_0 \in (0, \bar{\Delta})$ and $\eta \in [0, \frac{1}{4})$; initially $D^0 = I$ and $d^0 = -\nabla f\left(x^0\right)$, $\Delta_{min} = 10^{-5}$, $\gamma = 50$ and $\gamma_{max} = 10^3$.

While: x^k is unsatisfactory (i.e. $\left\|\nabla f(x^k)_{\mathcal{U}}\right\| \geq \varepsilon$?) **do**

Step 1: Obtain d^k by *approximately* solving the problem

$$\min_d m_k(d) := f\left(x^k\right) + \max\{\left((g^k)^T d, (g^{k-1})^T d\right\}\right) + \frac{1}{2} d^T B^k d$$
$$\text{Subject to } \|d\| \leq \Delta^k. \tag{5}$$

Step 2: Calculate $\rho_k := \frac{f\left(x^k\right) - f\left(x^k + d^k\right)}{m_k(0) - m_k\left(d^k\right)}$.

Step 3: **If:** $\rho_k < \frac{1}{16}$

$\Delta^{k+1} = \max\{\frac{1}{4}\left\|d^k\right\|, \Delta_{min}\}$ and $\gamma = \min\{(1.1) * \gamma, \gamma_{max}\}$.

else

If $\rho_k > \frac{3}{4}$ and $\left\|d^k\right\| = \Delta^k$ (*)

$\Delta^{k+1} = \min\left(4\Delta^k, \bar{\Delta}\right)$

else

$\Delta^{k+1} = \Delta^k$

end

end

If $\rho^k > \eta$

$x^{k+1} = x^k + d^k$

else

$x^{k+1} = x^k$

end

Step 4: **If** $\rho^k > \eta$ calculate

- Let $s^k = x^{k+1} - x^k$ and $y^k = g^{k+1} - g^k$ and
- Call $\left[d^{k+1}_{BFGS}, D^{k+1}\right] = BFGS\left(s^k, y^k, D^k\right)$.
- Call $[z^{k+1}, U^{k+1}, \lambda^{k+1}, V^{k+1}, \mu^{k+1}] = P_{\mathcal{V}}(x^k, D^k)$
  (where $u_i \in U^{k+1}$ and $v_i \in V^{k+1}$) and let

$$B^k := \sum_{i=1}^{n-m}\left(\lambda^{k+1}\right)^{-1} u_i u_i^T + \gamma \sum_{i=1}^{m} v_i v_i^T.$$

End While

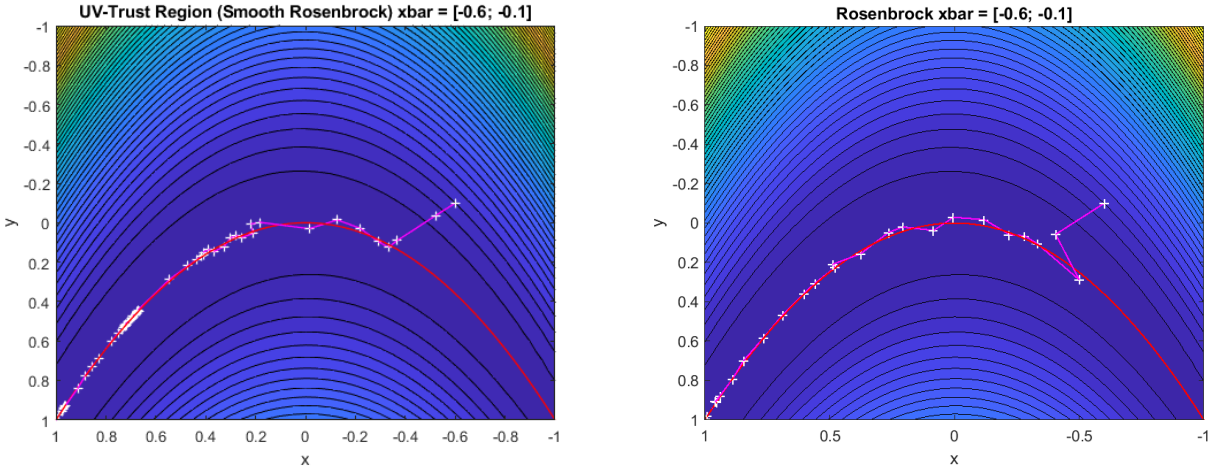
This modification in theory will allow the method to take longer steps and travel along the valley. By applying the spectral decomposition to the Hessian approximation B the matrix can be expressed in terms of the $\mathcal{VU}$ space:

$$B^k := \sum_{i=1}^{n-m} \left(\lambda^{k+1}\right)^{-1} u_i u_i^T + \gamma \sum_{i=1}^{m} v_i v_i^T$$

where λ^{k+1}, u and v are derived from the inverse Hessian eigendecomposition. This approximation of B^k allows for the direction component in the $\mathcal{V}$ space to be scaled by a factor γ which can be reduced to avoid collisions with the sides of the valley and hopefully allow longer steps to be taken. The update can be thought of as choosing the standard Trust Region direction to step in and subtracting away some of the $\mathcal{V}$ space component before taking a step.

The basic idea behind the philosophy of Trust Region methods is based on the presumption that the model (5) may be made to fit the function f locally to an arbitrary degree of accuracy (if we restrict it to a small enough ball). The idea of the model at the time was to: make a crude approximation of the directional derivative; approximate the Hessian along the $\mathcal{U}$ space as the inverse of the inverse Hessian approximation D^k; and attempt to control the amount of curvature along the $\mathcal{V}$ space (to avoid the infinite jump which happens at the "kink").

As a benchmark, the Trust Region method is tested on smooth Rosenbrock functions first and then compared to the classic "Dog-Leg" strategy (see appendix A) used for standard smooth Trust Region methods discussed in section 4. Both methods are then be tested on the non-smooth Rosenbrock function.



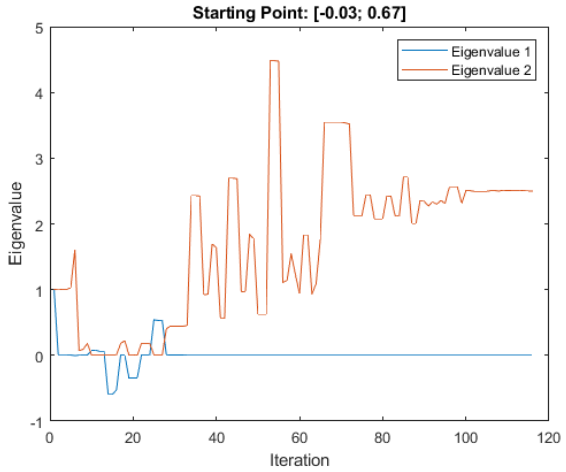
(a) Smooth Rosenbrock with VU - Trust Region

(b) Smooth Rosenbrock "Dog-Leg" Trust Region

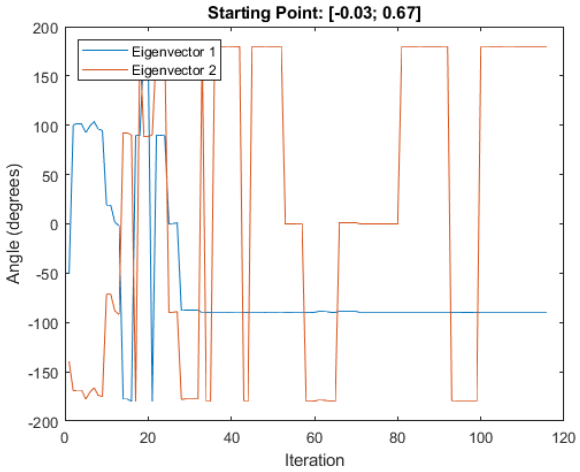
Figure 11

In this case, the $\mathcal{UV}$ Trust Region method is competitive time wise. It only slowed down twice in its decent which lead to approximately 3 times the number of iterations (116 for the $\mathcal{UV}$ Trust Region and 37 for the "Dog-Leg" Trust Region).

The behaviour of the eigenvalues and eigenvectors for the $\mathcal{UV}$ Trust Region was found to be consistent with that observed before, see Figure 12.



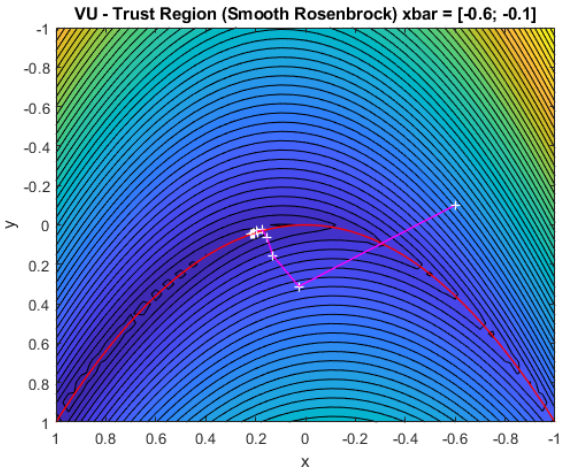
(a) Eigenvalues for D^k in the VU - Trust Region



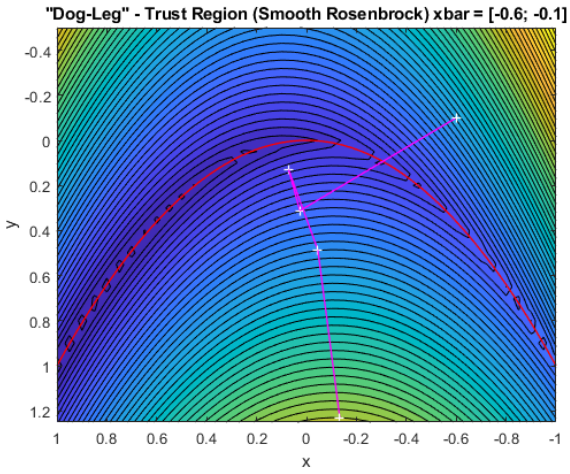
(b) Eigenvectors for D^k in the VU - Trust Region

Figure 12

Both methods were then tested on the non-smooth Rosenbrock function and their performance examined.



(a) VU-Trust Region (Non-smooth Rosenbrock)



(b) "Dog-Leg" Trust Region (Non-smooth Rosenbrock)

Figure 13

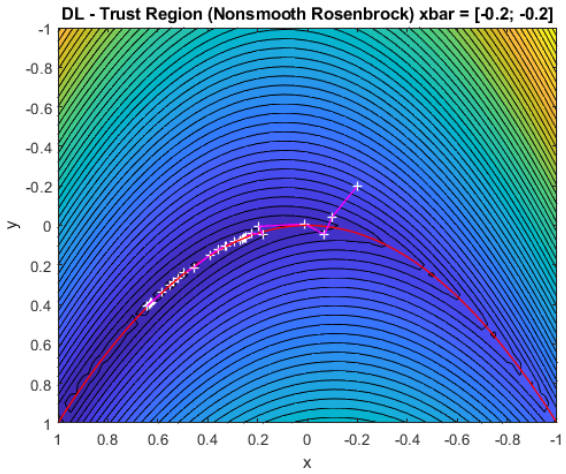


Figure 14: Trust Region without the "Dog-Leg" (Non-smooth Rosenbrock)

The $\mathcal{VU}$ Trust Region managed to locate the smooth manifold after several iterations, but δ then decreased to around 10^{-5} and remained constant. This appeared to be a failure in the Trust Region model chosen. The directions generated by the model provided only small relative descent and so ρ^k remained small. This resulted in a loop of reductions in δ^k without substantial model improvement and caused method to stall at the "kink" of the non-smooth function. The quadratic model clearly did not depict the $\mathcal{V}$ shape sufficiently to allow adequate descent. A remedy would be to collect additional past gradient information for the purpose of approximating the directional derivative more accurately. This would mimic "bundle methods" which evidence suggests that such strategies work for non-smooth optimisation. Unfortunately we did not have time to implement this approach. On the other hand, the classical "Dog-Leg" Trust Region failed dramatically and pathologically in a manner that was far more difficult to understand. The best explanation we have is that D^k became negative-indefinite while crossing the discontinuity, causing the "Dog-Leg" strategy to fail badly. This hypothesis was tested by running the same algorithm without the "Dog-Leg," but with a standard quadratic model direction finding step. This greatly improved performance, but again ultimately stalled as the model failed to provide adequate descent directions, see Figure 14.

Conclusion

This report considered the performance of the BFGS and Trust Region methods on non-smooth functions. It was observed that both methods tended to locate the valley of the non-smooth Rosenbrock function under consideration quickly and attempted to follow it closely towards the

minimum. The valley was found to cause both algorithms to encounter issues and modifications were proposed to address both of their limitations. The BFGS method was modified with a non-monotone line search and $\mathcal{V}$ space projection to step away from the valley and correct the following search direction. This allowed the method to travel towards the minimum at a distance from the valley on which the performance was better. The Trust Region modification corrected the direction of the quadratic model to prevent it from colliding with the narrow sides of the valley and allowed it to take larger steps. The optimisation of non-smooth functions still remains an under-researched area of mathematics. While discoveries were made in this report there is still so much more to be discovered and developed in this area.

Appendix A

The Dog-Leg

A standard way to generate a descent direction when a function is smooth uses the so called dog-leg approximation. This method tries to approximate the solution of the trust region sub problem as a function of the trust region size Δ using only the information provided by the input data of the basic quadric model. That is we let $d(\Delta)$ solve exactly the problem

$$\min_d m_k(d) := f\left(x^k\right) + \nabla f\left(x^k\right)^T d + \frac{1}{2} d^T B_k d$$
$$\text{Subject to} \quad \|d\| \leq \Delta$$

Clearly the better strategy would be to take

$$d^B := -B_k^{-1} \nabla f\left(x^k\right) \quad \text{when} \quad \left\|d^B\right\| \leq \Delta^k$$

if possible. In this case when $B_k = \nabla^2 f\left(x^k\right)$ we would expect superlinear convergence. We will assume B_k is positive definite (which is the case for DFP and BFGS approximations of the Hessian). First consider minimizing $\tau \mapsto m_k(\tau d_s)$ along the steepest descent direction $d_s = -g$ where $g = \nabla f\left(x^k\right)$. Then

$$\frac{d}{d\tau} m_k(\tau d_s) = \nabla f\left(x^k\right)^T d_s + \tau \left(d_s\right)^T B_k d_s = 0$$
$$\text{and so} \quad \tau = -\frac{\nabla f\left(x^k\right)^T d_s}{\left(d_s\right)^T B_k d_s} = \frac{g^T g}{g^T B_k g}.$$

Thus we would take a step given by

$$d^U = -\tau g = -\frac{g^T g}{g^T B_k g} g.$$

The dog-leg choose d tries to approximate the optimal trajectory (as a function of Δ) by a piecewise linear approximation. That is if we let $d(\Delta)$ solve exactly the problem

$$\begin{aligned} \min_d m_k(d) &:= f\left(x^k\right) + \nabla f\left(x^k\right)^T d + \frac{1}{2} d^T B_k d \\ \text{Subject to} \quad & \|d\| \leq \Delta \end{aligned}$$

then this would be a curve in $\mathbb{R}^n$. We can approximate this curve using the so called dog-leg

$$d(\tau) := \begin{cases} \tau d^U & \text{if} \quad 0 \leq \tau \leq 1 \\ d^U + (\tau - 1)\left(d^B - d^U\right) & \text{for } 1 \leq \tau \leq 2 \end{cases}.$$

The following show that $d(\tau)$ will intersect the trust region boundary at exactly one point if $\left\|d^B\right\| \geq \Delta$ and nowhere otherwise. Thus we should take

$$d = d^B \quad \text{when} \quad \left\|d^B\right\| < \Delta.$$

Otherwise the intersection of the dog-leg with the trust region boundary. We have $0 < \tau \leq 1$ when $\left\|d^U\right\| \geq \Delta$ and so

$$\tau = \frac{\Delta}{\|d^U\|} \quad \text{with} \quad d = \tau d^U.$$

When $1 < \tau \leq 2$ we may find τ by solving the scalar quadratic equation

$$\left\|d^U + (\tau - 1)\left(d^B - d^U\right)\right\|^2 = \Delta^2.$$

References

[1] Lewis, A. S. and Overton, (2013) M. L., *Nonsmooth optimization via quasi-Newton methods*, Math. Program., **141**, 1-2, Ser. A, pp; 135–163, ISSN: 0025-5610, 10.1007/s10107-012-0514-2

[2] Lewis, A. S. and Zhang, S.,(2015) *Nonsmoothness and a variable metric method*, J. Optim. Theory Appl., VOLUME = **165** (1), pp: 151–171, ISSN: 0022-3239, DOI = 10.1007/s10957-014-0622-7.

[3] Guo, J. and Lewis, A. S., (2018) *Nonsmooth variants of Powell's BFGS convergence theorem*, SIAM J. Optim., **28**(2), pp: 1301–1311, ISSN: 1052-6234, DOI = 10.1137/17M1121883.

[4] Grippo, L. and Lampariello, F. and Lucidi, S., (1986) *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., **23** (4), pp: 707–716, ISSN: 0036-1429, DOI = 10.1137/0723046.

[5] A. R Conn ; Nicholas I. M Gould; Ph. L Toint (2000), Trust-region methods, SIAM, MPS-SIAM series on optimization.

[6] Yu, Jin, Vishwanathan, S. V. N., Günter, Simon and Schraudolph, N. N., (2010) *A quasi-Newton approach to nonsmooth convex optimization problems in machine learning*, J. Mach. Learn. Res., **11**, pp: 1145–1200, ISSN : 1532-4435,

[7] C. Lemarèchal, F. Oustry and C. Sagastizabal (1999), *The $\mathcal{U}$-Lagrangian of a Convex Function,* Trans Amer. Math. Soc., **352**(2), pp: 711-729.

[8] Hare, W. L., (2006), *Functions and sets of smooth substructure: relationships and examples*, Comput. Optim. Appl., 33, **2-3**, pp: 249–270, ISSN:0926-6003, DOI 10.1007/s10589-005-3059-4,

[9] Mifflin, R. and Sagastizábal, C., (2005) *A $\mathcal{VU}$-algorithm for convex minimization*, Math. Program., 104, (2-3, Ser. B), pp: 583–608, ISSN: 0025-5610, DOI 10.1007/s10107-005-0630-3,