Efficient Trust Region Subproblem Algorithms

by

Heng Ye

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Combinatorics and Optimization

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis.

I understand that my thesis may be made available to the public.

Abstract

The *Trust Region Subproblem* (TRS) is the problem of minimizing a quadratic (possibly nonconvex) function over a sphere. It is the main step of the trust region method for unconstrained optimization problems. Two cases may cause numerical difficulties in solving the TRS, i.e., (i) the so-called hard case and (ii) having a large trust region radius. In this thesis we give the optimality characteristics of the TRS and review the major current algorithms. Then we introduce some techniques to solve the TRS efficiently for the two difficult cases. A shift and deflation technique avoids the hard case;, and a scaling can adjust the value of the trust region radius. In addition, we illustrate other improvements for the TRS algorithm, including: rotation, approximate eigenvalue calculations, and inverse polynomial interpolation. We also introduce a warm start approach and include a new treatment for the hard case for the trust region method. Sensitivity analysis is provided to show that the optimal objective value for the TRS is stable with respect to the trust region radius in both the easy and hard cases. Finally, numerical experiments are provided to show the performance of all the improvements.

Acknowledgements

I would like to thank my supervisor, Prof. Henry Wolkowicz for his guidance and support during my graduate studies. I would like to thank my readers Prof. Stephen Vavasis and Prof. Thomas Coleman for their time. I would also like to thank Dr. Marielba Rojas for providing and revising the LSTRS software.

Table of Contents

List of Tables

List of Figures

1 Introduction

We are concerned with the following quadratic minimization problem, called the *trust region subproblem*

$$(TRS) \qquad \begin{array}{rll} q^* = & \min & q(x) := x^T Ax - 2a^T x \\ & \text{s.t.} & \|x\|^2 \leq s^2. \end{array} \tag{1}$$

Without loss of generality, we assume that $A \in \mathbb{R}^{n\times n}$ is a symmetric matrix,[1] $a \in \mathbb{R}^n, s \in \mathbb{R}_{++}$ are given, and $x \in \mathbb{R}^n$ is the unknown variable. Here, $\|x\| := \sqrt{x^T x}$ is the Euclidean ℓ_2 norm.

Solving the TRS is the main step of the trust region method for unconstrained minimization. Precisely, it gives the step of each iteration in the trust region method. Moreover, it has many other applications in e.g., regularization, sequential quadratic programming, and discrete optimization.

One of the main contributions of this thesis is a modified approach for solving the TRS that involves a simple scaling. We analyze how different scalars can affect the so-called *RW algorithm* [**?**], and suggest several methods to select the best scalar. Moreover, the scalar can also be viewed as another parameter, and this gives rise to a new approach for solving the TRS. Besides the scaling, we also study the shift and deflation step introduced in [**?**], which enables us to obtain an equivalent TRS with a positive semi-definite matrix A if the so-called *hard case* holds. Therefore, it can be solved efficiently using preconditioned conjugate gradients. The solution to the original TRS can then be easily recovered by taking a *primal step* to the boundary.

Moreover, new techniques are also introduced to improve the efficiency of the algorithm, including rotation, inverse polynomial interpolation and approximate eigenvalue calculations. The performance of such techniques are all illustrated by numerical tests. Furthermore, we provide a sensitivity analysis for TRS, and show that, under certain assumptions, the optimal value of the objective function is stable with respect to perturbations in the trust region radius.

We conclude by showing how our revised algorithm behaves within the trust region method; and, we also introduce two modifications to the method: (i) updating the radius in a different way if the hard case occurs; and, (ii) using warm starts when a step is not taken. These modifications not only improve the stability, but they also improve the efficiency of the trust region method.

1.1 Outline of Results

We continue in Section **??** and give a brief review of the optimality conditions of the TRS. Then, in Section **??** we provide a survey of many of the current algorithms: the MS algorithm, [**?**], the GLTR algorithm [**?**], the SSM algorithm [**?**], the Gould-Robinson-Thorne algorithm [**?**], and the RW algorithm [**?**, **?**]. In Section **??**, we introduce the concept of scaling and how it can apply to the RW algorithm. In Section **??**, we show how shift and deflation are applied to the algorithm to prevent the hard case (case 2). In Section **??**, other techniques and sensitivity analysis are provided, followed by the application of the algorithm on unconstrained optimization in Section **??**, and the numerical tests in Section **??**.

2 Optimality Conditions

The optimality conditions for the TRS are described by the following theorem:

[1] If A is not symmetric, then we can always replace A by $(A + A^T)/2$; and, an equivalent problem with a symmetric matrix is obtained.

Theorem 2.1. *x^* is a solution to TRS if and only if for some (Lagrange multiplier) $\lambda^* \in \mathbb{R}$, we have*

$$\begin{array}{cc} \left.\begin{array}{c} (A-\lambda^* I)x^* = a \\ A-\lambda^* I \succeq 0, \lambda^* \leq 0 \end{array}\right\} & \textit{dual feasibility} \\ \|x^*\| \leq s & \textit{primal feasibility} \\ \lambda^*(s-\|x^*\|)=0. & \textit{complementary slackness} \end{array} \tag{2}$$

where $A-\lambda^* I \succeq 0$ denotes that $A-\lambda^* I$ is positive semi-definite. ■

Throughout the thesis, we define that (x^*, λ^*) solves the TRS if and only if x^* and λ^* satisfy the optimality conditions.

These optimality conditions were developed by Gay [**?**] and Moré and Sorensen [**?**]. We presented it above in (**??**) using the modern primal-dual paradigm, see e.g., [**?**, **?**, **?**] and [**?**, **?**]. However, it should be mentioned that an earlier result related to the optimality condition of the TRS is the S-lemma developed by Yakubovich in 1971, [**?**]. This lemma was originally motivated by showing whether a quadratic (in)equality is a consequence of other (in)equalities. It has extensive applications in quadratic and semi-definite optimization, convex geometry and linear algebra. It can be shown that the optimality conditions of the TRS can also be derived from the S-lemma, see e.g., [**?**].

Lemma 2.1. *(S-lemma [**?**]) Let f, g: $\mathbb{R}^n \to \mathbb{R}$ be quadratic functions and suppose that there is an $\bar{x} \in \mathbb{R}^n$ such that $g(\bar{x}) < 0$. Then the following two statements are equivalent.*

(i) There is no $x \in \mathbb{R}^n$ such that

$$f(x) < 0, \quad g(x) \leq 0.$$

(ii) There is a number $\lambda \leq 0$ such that

$$f(x) - \lambda g(x) \geq 0 \quad \forall x \in \mathbb{R}^n.$$

■

For completeness, we now show how to use the S-lemma to prove the optimality conditions of the TRS.

Proof: (of Theorem ?? using S-lemma)

Let

$$q^* := q(x^*), \quad f(x) := q(x) - q^*, \quad g(x) := x^T x - s^2, \quad s > 0.$$

If x^* is the minimizer of the TRS, then we know that $\|x^*\| \leq s$. Since x^* is the minimizer, we get $q(x) \geq q(x^*) = q^*$, and so we conclude $g(x) \leq 0 \implies f(x) \geq 0$. Therefore, there is no $x \in \mathbb{R}^n$ such that $f(x) < 0$ and $g(x) \leq 0$. Now, according to the S-lemma, $\exists \lambda^* \leq 0$ such that

$$L(x) = f(x) - \lambda^* g(x) \geq 0, \quad \forall x \in \mathbb{R}^n.$$

Since $L(x)$ is bounded below, we get $\nabla^2 L(x) \succeq 0$, and we have

$$A - \lambda^* I \succeq 0.$$

If $\|x^*\| = s$, then

$$L(x^*) = q(x^*) - q^* - \lambda^*(\|x^*\|^2 - s^2) = 0.$$

In addition, since $L(x) \geq 0 \forall x \in \mathbb{R}^n$, we have $x^* \in \operatorname{argmin} L(x)$, and $\nabla L(x^*) = 0$, which gives

$$(A - \lambda^* I)x = a.$$

If $\|x^*\| < s$, then

$$L(x^*) = q(x^*) - q^* - \lambda^*(\|x^*\|^2 - s^2) = \lambda^*(s^2 - \|x^*\|^2).$$

Since $\lambda^* \leq 0$, $s^2 - \|x^*\|^2 > 0$, we have $L(x^*) \leq 0$. However, $L(x) \geq 0 \quad \forall x \in \mathbb{R}^n$, then we know $L(x^*) = 0$ and

$$\lambda^* = 0.$$

Similarly as in the case of $\|x^*\| = s$, since x^* minimizes $L(x)$, we conclude that $\nabla L(x^*) = 0$. Then, we have

$$(A - \lambda^* I)x = a.$$

This completes the proof of necessity of the optimality conditions.
For sufficiency, suppose that x^* and λ^* satisfy all the optimality conditions in (**??**). Let

$$\begin{aligned} L(x) &= f(x) - \lambda^* g(x) \\ &= q(x) - q^* - \lambda^*(x^T x - s^2). \end{aligned}$$

Since

$$\nabla L(x^*) = (A - \lambda^*)x^* - a = 0,$$

$$\nabla^2 L(x^*) = A - \lambda^* I \succeq 0,$$

we know

$$x^* \in \operatorname{argmin} L(x).$$

Also, because $\lambda^*(\|x^*\|^2 - s^2) = 0$, we conclude that $\forall x \in \mathbb{R}^n$,

$$L(x) \geq L(x^*) = q(x^*) - q^* - \lambda^*(\|x^*\|^2 - s^2) = 0.$$

According to S-lemma, we know that if $g(x) \leq 0$ (i.e., $x^T x \leq s^2$), then $f(x) \geq 0$, which means $q(x) \geq q(x^*)$. Therefore, x^* solves the TRS. ■

Even though the optimality conditions are well known, standard algorithms have numerical difficulties solving the TRS when the hard case occurs; we now describe this case. Let $\lambda_{\min}(A)$ be the smallest eigenvalue of A. From the optimality conditions, we know that $\lambda^* \leq \lambda_{\min}(A)$. Throughout the thesis, we define $x(\lambda) = (A - \lambda I)^\dagger a$, where † denotes the Moore-Penrose generalized inverse, i.e.,

$$x(\lambda) = \begin{cases} (A - \lambda I)^{-1} a & \text{if } \lambda < \lambda_{\min}(A) \\ (A - \lambda I)^\dagger a = \begin{array}{ll} \operatorname{argmin} & \|x\| \\ \text{s.t.} & x \in \operatorname{argmin} \|(A - \lambda I)x - a\| \end{array} & \text{if } \lambda = \lambda_{\min}(A). \end{cases}$$

Note that the optimality conditions imply that $(A - \lambda^* I)x = a$ is always consistent, so that $x \in \operatorname{argmin} \|(A - \lambda I)x - a\|$ can be replaced by $(A - \lambda I)x = a$ in the case that $\lambda^* = \lambda_{\min}(A)$. Let $A = Q\Lambda Q^T$ be the spectral decomposition of A, then we can see that when $\lambda < \lambda_{\min}(A)$,

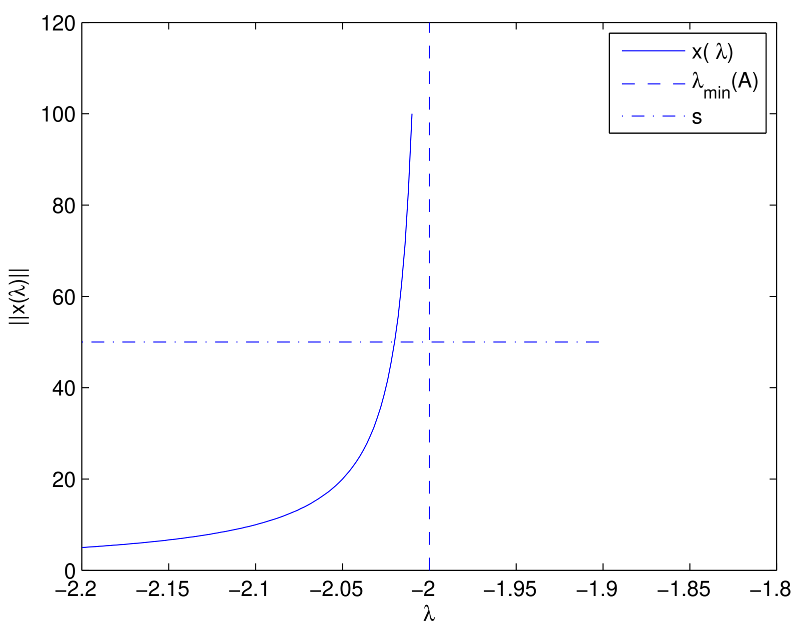
$$\begin{aligned} \|x(\lambda)\|^2 &= \|(Q\Lambda Q^T - \lambda I)^{-1} a\|^2 \\ &= \|Q(\Lambda - \lambda I)^{-1} Q^T a\|^2 \\ &= \|(\Lambda - \lambda I)^{-1} Q^T a\|^2 \\ &= \sum_{i=1}^{n} \frac{\gamma_i^2}{(\lambda_i - \lambda)^2}, \end{aligned} \tag{3}$$

where λ_i is the ith smallest eigenvalue of A, and γ_i is the corresponding component of $Q^T a$. Therefore, when $\lambda < \lambda_{\min}$, $\forall i = 1, ..., n$, we have $\lambda_i - \lambda > 0$, and so $\|x(\lambda)\| = \|(A - \lambda I)^{-1} a\|$ is a monotonically increasing function for $\lambda \in (-\infty, \lambda_{\min}(A))$. When $\exists i$ such that $\lambda_i = \lambda_{\min}$ and $\gamma_i = Q(:, i)^T a \neq 0$, i.e., at least one of the γ_i corresponding to the smallest eigenvalue is non-zero, we have $\|x(\lambda)\| \to +\infty$, as $\lambda \to \lambda_{\min}(A)$. Therefore, there is a unique λ^* satisfying the optimality conditions (see Figure **??**):

$$\|x(\lambda^*)\| = \|(A - \lambda^* I)^{-1} a\| = s, \quad \lambda^* < \lambda_{\min}(A).$$

This is called the *easy case*.

Figure 1: $\|x(\lambda)\|$ in Easy Case



However, if $a \in \mathcal{R}(A - \lambda_{\min}(A)I)$ (the orthogonal complement to the eigenspace for $\lambda_{\min}$), then there exists $x(\lambda_{\min}(A))$ such that $(A - \lambda_{\min}(A)I)x(\lambda_{\min}(A)) = a$. If $\|x(\lambda_{\min}(A))\| > s$, we know that $\lambda^* < \lambda_{\min}(A)$ because $\|x(\lambda)\|$ is strictly increasing when $\lambda < \lambda_{\min}(A)$ and $x(\lambda^*) = s < \|x(\lambda_{\min}(A))\|$. This is called the *hard case (case 1)*.

On the other hand, it is possible that $\|x(\lambda_{\min}(A))\| \leq s$ (see Figure **??**). In such case, we need to let $\lambda^* = \lambda_{\min}(A)$. When $\lambda^* = \lambda_{\min}(A) = 0$ or $\|x(\lambda^*)\| = s$, then complementary slackness holds with $x^* = x(\lambda^*)$. However, if neither of the two conditions holds, then we need to find $z \in \mathcal{N}(A - \lambda^* I)$ (which is not unique) such that $\|x(\lambda^*) + z\| = s$. Because $z \in \mathcal{N}(A - \lambda^* I)$, we have $(A - \lambda^* I)(x(\lambda^*) + z) = a + 0 = a$, and therefore the optimality conditions are satisfied with $x^* = x(\lambda^*) + z$. This is called the *hard case (case 2)*. This description of the hard case appears in [**?**]. The different cases amplify on the definition given in e.g., [**?**, **?**] where the hard case is characterized by the condition $a \in \mathcal{R}(A - \lambda_{\min}(A)I)$ and $\|x(\lambda_{\min}(A))\| \leq s$, i.e., only hard case (case 2) is considered hard case.

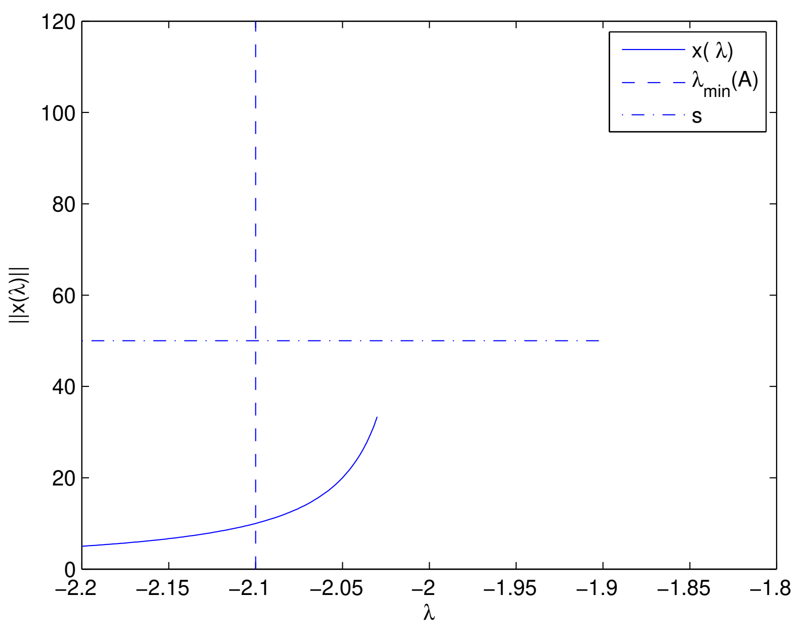
Hard case problems are often considered more difficult to solve in the literature. The hard case (case 2) results in the solution to the TRS to be non-unique, i.e., we have an ill-posed problem in the Tikhonov sense, [**?**]. Moreover, many algorithms are trying to find the value of λ^* such that

$$\lambda^* \leq 0, \quad \lambda^* < \lambda_{\min}(A), \quad \|(A - \lambda^* I)^{-1} a\| = s. \tag{4}$$

If the hard case (case 1) occurs, λ^* can still be found by standard root finding algorithms like Newton's method. However, if the hard case (case 2) occurs, $\lambda^* = \lambda_{\min}(A)$, and therefore $(A - \lambda^* I)^{-1} a$ is indeed not defined. Therefore, for most algorithms for solving the TRS, some other techniques usually have to be exploited to handle the hard case (case 2). More details will be given in Section **??**.

However, in our algorithm we exploit the special structure of the hard case (case 2) and, in fact, show that the *hard case problems are usually the easier problems* to solve. This result will be shown in Section **??** and Section **??**.

Figure 2: $\|x(\lambda)\|$ in Hard Case (Case 2)



3 Survey of Several Current Algorithms

In spite of its simple structure, there have been an enormous number of methods proposed that solve the TRS efficiently. Many of the algorithms try to solve the problem iteratively, since no explicit solution is known. Moreover, though the existence of the hard case has been well studied, not all the methods are able to handle it properly. We now give a brief introduction of the major algorithms.

In 1981, Gay (see [**?**]) proposed the optimality conditions for the TRS and used an iterative algorithm to solve it based on Newton's method. Newton's method is safeguarded by an upper bound and a lower bound in each iteration in order to maintain positive definiteness of the Hessian of the Lagrangian, and so the algorithm converges to the solution. Moreover, the secular equation is introduced to achieve fast convergence. In the paper, it is also discussed that the hard case may occur, which would cause the algorithm to fail. However, it does not treat the hard case efficiently.

In 1983, More and Sorensen (see [**?**]) modified Gay's algorithm and proposed the so-called *MS Algorithm*, which remains one of the classic methods for solving the TRS. Special properties of the Cholesky factorization are exploited in each iteration to make the algorithm more efficient. Most importantly, the algorithm is able to detect a *possible* hard case and take advantage of it. This is done by making λ approach $\lambda_{\min}(A)$ and take a primal step to the boundary in each iteration if $\|x(\lambda)\| < s$. i.e., find $z \in \mathcal{N}(A - \lambda_{\min}(A)I)$ such that $\|x(\lambda) + z\| = s$.

The previous two methods both involve matrix factorization in each iteration, and hence can not take advantage of sparse or structured matrices. In 1994, Rendl and Wolkowicz (see [**?**]) and Sorensen (see [**?**]) used different approaches to reformulate the problem to an equivalent parametric eigenvalue problem, and therefore sparsity can be exploited if Lanczos type methods (see [**?**] and [**?**]) are used to compute the eigenvalues and eigenvectors. Precisely, the matrix A is only used as a multiplication operator on a vector, while no factorization is needed.

In 1993, Ben-Tal and Teboulle (see [**?**]) used the *double duality* (second dual) approach to derive that the TRS is always equivalent to a convex optimization problem. This convex problem is a special case of the TRS where the matrix A is diagonal. Moreover, the results can be extended to quadratic minimization problems with two-sided (possibly indefinite) quadratic constraints.

In 1995, Tao and An (see [**?**]) applied the Difference of Convex Functions Algorithm (DCA) to solve the TRS. However, one of the big problems with this algorithm is that it cannot guarantee global convergence.

In 1999, Gould, Lucidi, Roma and Toint (see [**?**]) presented the *generalized Lanczos trust region method* (GLTR). It is based on the truncated Lanczos method but uses the information of the whole Krylov subspace generated in the previous iterations. This method can also exploit sparsity, but it fails to handle the hard case.

Also, Rojas, Santos and Sorensen (see [**?**]) presented the Large-Scale Trust-Region Subproblem method (LSTRS), which is a revision of the Sorensen method (see [**?**]). The algorithm is matrix-free in the sense that the matrix is only used as a multiplication operator on a vector. Also, it uses a different interpolating scheme and introduces a unified iteration that includes the hard case.

In 2000, Hager (see [**?**]) proposed the *sequential subspace method* (SSM). There are two phases in this algorithm, where the first phase is very similar to the GLTR algorithm, which is only used to generate an initial guess. In the second phase, the problem is solved over a subspace which is adjusted in successive iterations. However, instead of the Krylov subspace, the algorithm constructs a subspace with dimension 4, so that in each iteration the problem is easy to solve and global convergence is also guaranteed (see [**?**]).

In 2001, Ye and Zhang (see [?]) studied the so-called *extended trust region subproblem.* It is the minimization of a quadratic function subject to two quadratic inequalities. It shows that the semi-definite programming relaxations of the extended trust region subproblems are exact in some special cases (which include the TRS), in the sense that the optimal values of the relaxation problems are equal to the original problems. As a consequence, polynomial time procedures can be obtained to solve these optimization problems.

In 2009, Gould, Robinson and Thorne (see [?]) made another revision to the MS Algorithm. Instead of Newton's method, which is only a second order model, the new algorithm uses high-order polynomial approximation and inverse interpolation to improve the accuracy of each iteration. The hard case can also be solved efficiently.

Erway, Gill and Griffin (see [?]) also proposed the *Phased-SSM* algorithm , which is based on the SSM algorithm. It is different from the SSM by adding an inexpensive estimate of the smallest eigenvalue of the matrix and a parameter to control the tradeoff number of iterations and number of matrix-vector multiplications. Moreover, a regularized Newton's method generates an accelerator direction in the low-dimensional subspace used in the second phase.

Also, in 2011, Lampe, Rojas, Sorensen and Voss made another modification to the LSTRS (see [?]). Improvements are mainly achieved by using the nonlinear Arnoldi method to solve the eigenvalue problem. This modification is able to make use of all the information of the eigenvalues and eigenvectors from the previous iterations. Numerically, this modification is shown to significantly accelerate the LSTRS.

In the following sections, we provide a detailed review of the MS algorithm, the GLTR algorithm, the SSM algorithm, the Gould-Robinson-Thorne algorithm, and the RW algorithm.

3.1 The MS Algorithm

Generally speaking, this algorithm (see [?]) solves the optimality condition equation $\|x(\lambda)\| = \|(A-\lambda I)^{-1}a\| = s$ by Newton's method, unless the hard case is indicated by the current minimum estimate lying in the interior of the trust region. In each iteration, a Cholesky factorization is used to calculate Newton step, which is also the main cost of this algorithm. In addition, a safeguarding scheme is exploited to ensure the conditions $A-\lambda I \succeq 0$ (i.e., $\lambda \leq \lambda_{\min}(A)$) and $\lambda \leq 0$ are satisfied. When the hard case is detected, (i.e., $a \in \mathcal{R}(A-\lambda_{\min}(A)I)$ and $(A-\lambda_{\min}(A)I)^{\dagger}a < s$), a primal step to the boundary is taken.

3.1.1 The Easy Case

From (**??**), we know that

$$\|x(\lambda)\|^2 = \sum_{i=1}^{n} \frac{\gamma_i^2}{(\lambda_i-\lambda)^2}.$$

This function is an extremely non-linear function with respect to λ, especially when λ is close to $\lambda_{\min}(A)$, and results in low convergence rate and poor performance when using Newton's method. To fix this problem, we can replace the function by $\frac{1}{\|x(\lambda)\|}$ and alternatively solve the equivalent equation

$$\Psi(\lambda) = \frac{1}{\|x(\lambda)\|} - \frac{1}{s} = 0, \tag{5}$$

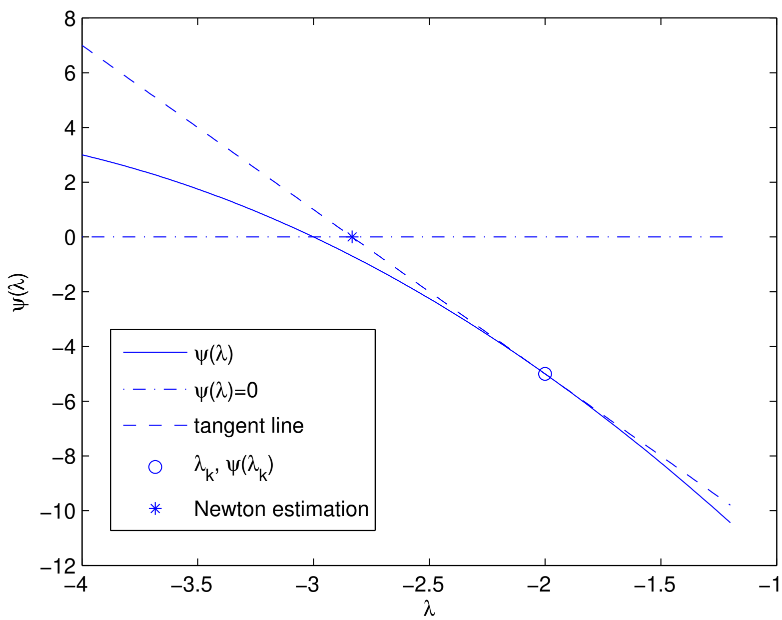
which is well known as the *secular equation.* The details of the algorithm is shown in Algorithm **??**:

Algorithm 3.1. ***Main Steps of the MS Algorithm***

Suppose $\lambda_k < 0$ and $\lambda_k < \lambda_{\min}(A)$; perform the following steps until the algorithm terminates:

1. *Factorize $A - \lambda_k I = LL^T$ by Cholesky factorization.*
2. *Calculate x such that $LL^T x = a$.*
3. *Calculate y such that $Ly = x$.*
4. *Update $\lambda_{k+1} = \lambda_k - \Psi(\lambda_k)/\Psi'(\lambda_k) = \lambda_k - (\frac{\|x\|}{\|y\|})^2(\frac{\|x\|-s}{s})$.*

Figure 3: Newton Step in MS Algorithm



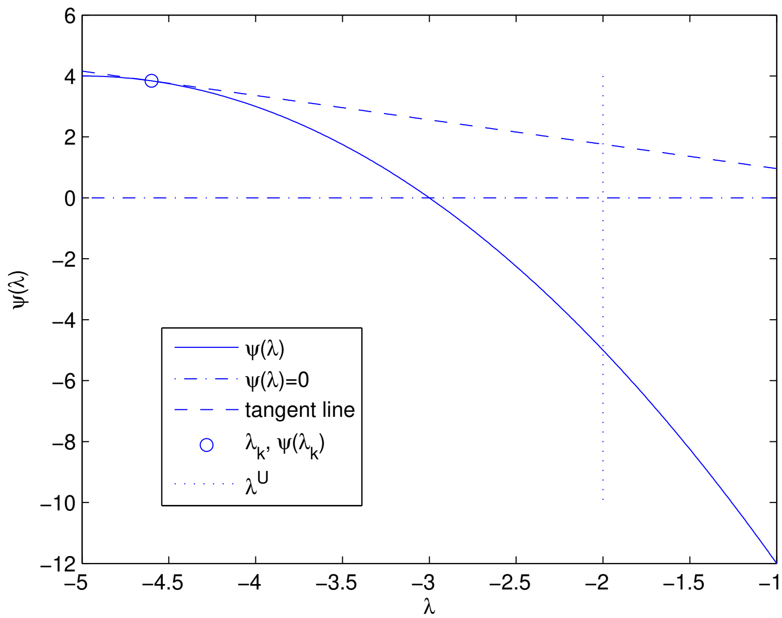
From the graph of the function $\Psi(\lambda)$, we know that when λ_k is in the interval $(\lambda^*, \min(0, \lambda_{\min}(A)))$, then Newton's method monotonically converge to λ^* (see [?] and Figure **??**). However, when λ_k is less than λ^*, the next iteration in Newton's method has poor performance (see Figure **??**). Therefore, the following safeguarding scheme in Algorithm **??** is used:

Algorithm 3.2. ***Safeguarding Scheme***

Initialize $\lambda^L = -\infty$, $\lambda^U = \min(0, \lambda_{\min}(A))$.

1. *If $\Psi(\lambda^U) > 0$, then the minimum lies in the interior or the hard case occurs, stop.*
2. *If $\Psi(\lambda_k) < 0$ and $\lambda_k < \lambda^U$, replace λ^U by λ_k.*
3. *If $\Psi(\lambda_k) > 0$ and $\lambda_k > \lambda^L$, replace λ^L by λ_k.*
4. *If $\lambda_k > \lambda^U$, replace λ_k by λ^U; if $\lambda_k < \lambda^L$, replace λ_k by λ^L.*

Figure 4: Safeguarding Newton Step in MS Algorithm



If $\lambda^U = 0$ and $\Psi(\lambda^U) > 0$, we know that $\lambda^* = 0$ and the minimum is in the interior. Therefore, $x^* = A^{-1}a$. If $\lambda^U = \lambda_{\min}(A)$ and $\Psi(\lambda^U) = \frac{1}{\|x(\lambda^U)\|} - \frac{1}{s} > 0$, i.e., the possible hard case occurs. How the hard case is handled is described in the following Section **??**. Except for these 2 situations, the algorithm converges to the optimal dual variable λ^*.

3.1.2 The Hard Case

The MS algorithm handles the hard case efficiently. As mentioned above, when a possible hard case is detected, we already have an x such that $\|x\| < s$ and $(A - \lambda_{\min}(A)I)x = a$. We need to find a $z \in \mathbb{R}^n$ such that $(A - \lambda_{\min}(A)I)z = 0$ and $\|x + z\| = s$. Since the main steps of the algorithm involve the dual variable λ, this is usually called *taking a primal step to the boundary*. Notice that because $x \in \mathcal{R}(A - \lambda_{\min}(A)I)$ and $z \in \mathcal{N}(A - \lambda_{\min}(A)I)$, we have $x \perp z$ and $\|x + z\| = \|x\| + \|z\|$. How we find z is illustrated by the following:

Theorem 3.1 ([**?**]). *Suppose*

$$A - \lambda I = LL^T, (A - \lambda I)x = a, \lambda \leq 0.$$

If

$$\|x + z\| = s, \|L^T z\|^2 \leq \sigma(\|L^T x\|^2 - \lambda s^2),$$

then

$$|q(x + z) - q(x^*)| \leq \sigma |q(x^*)|.$$

where σ is a scalar, and x^* is the optimal solution to the TRS.

■

Therefore, if we can find z such that $\|L^T z\|^2 \leq \sigma(\|L^T x\|^2 - \lambda s^2)$ with a σ sufficiently small, we know that $x + z$ is near optimal.

Even though the MS algorithm is able to solve the TRS in both the easy case and the hard case, there are some problems with the algorithm. First of all, in each iteration, a Cholesky factorization is used, which is too expensive for large-scale problems. In addition, sparsity cannot be exploited (even if the original matrix is sparse, the factorizations are generally dense due to fill-in). Secondly, if λ^* is less than but very close to $\min(0, \lambda_{\min}(A))$, we call it the *almost hard case*. If this occurs, the interval $(\lambda^*, \min(0, \lambda_{\min}(A)))$ is small so that it is extremely difficult to find a λ in this interval. At last, if $\lambda_{\min}(A) \leq 0$, when λ is approaching $\lambda_{\min}(A)$, the matrix $A - \lambda I$ becomes more and more ill-conditioned, which also causes computational difficulties.

3.2 The GLTR Algorithm

One of the difficulties in solving the TRS is that we need to search for x in $\mathbb{R}^n$, which might have large dimension. Many algorithms try to reduce the computation cost by obtaining an approximate solution in a smaller space (e.g., the Cauchy point method and the dogleg method, see [**?**]). Among all such methods, the generalized Lanczos trust region algorithm (see [**?**]) is remarkable. It solves the problem in a Krylov subspace and keeps expanding it until a satisfactory solution is obtained. In each iteration, it's equivalent to solving a TRS with a tridiagonal matrix, where the computation cost is comparatively low and the structure of the matrix can also be exploited.

In order to understand this method more clearly, we first introduce the *Steihaug-Toint algorithm*. It successively solves the following problem:

$$\begin{array}{ll} \min & q(x) = x^T A x - 2a^T x \\ s.t. & \|x\| \leq s \\ & x \in \mathcal{K}_k, \end{array} \tag{6}$$

where

$$\mathcal{K}_k = \text{span}\,\{a, Aa, A^2 a, ...A^{k-1}a\},$$

until the solution reaches the trust region boundary. Here, $\mathcal{K}_k$ is also widely known as the Krylov subspace generated by the matrix A and the vector a. However, as long as the solution touches or moves across the boundary, the algorithm terminates, even if the current iteration does not actually solve the problem. The GLTR can be viewed as an modification to the Steihaug-Toint method, so that it still works when the solution lies in the interior of the trust region.

The Lanczos method is one of the important methods used to generate a basis of the Krylov subspace and calculate the eigenvalues, see Algorithm **??**.

Algorithm 3.3. ***Lanczos Method***
Given a matrix A and a vector a, initialize $t_0 = a, w_{-1} = 0$. for $j = 0, 1, ..., k$, perform the following steps:

1. $\gamma_j = \|t_j\|$.
2. $q_j = t_j / \gamma_j$.
3. $\delta_j = q_j^T A q_j$.
4. $t_{j+1} = Aq_j - \delta_j q_j - \gamma_j q_{j-1}$.

It follows that (see [**?**])

$$\mathcal{K}_k = \text{span}\{q_0, q_1, ..., q_k\}$$

$$Q_k^T Q_k = I$$

$$AQ_k - Q_k T_k = \gamma_{k+1} w_{k+1} e_{k+1}^T$$

where $Q_k = [q_0, q_1, ..., q_k]$ and T_k is a tridiagonal matrix such that

$$T_k = \begin{pmatrix} \delta_0 & \gamma_1 & & & \\ \gamma_1 & \delta_1 & & & \\ & & \ldots & & \\ & & & \delta_{k-1} & \gamma_k \\ & & & \gamma_k & \delta_k \end{pmatrix}.$$

However, it's observed that such a basis and decomposition can also be obtained from the conjugate gradient method, Algorithm **??**:

Algorithm 3.4. ***Conjugate Gradient Method***
Given a matrix A and a vector a, initialize $g_0 = a$, $p_0 = -g_0$, and for $j = 0, 1, ..., k$, perform the following steps:

1. $\alpha_j = g_j^T g_j / p_j^T A p_j$.
2. $g_{j+1} = g_j + \alpha_j A p_j$.
3. $\beta_j = g_{j+1}^T gj + 1 / g_j^T g_j$.
4. $p_{j+1} = -g_{j+1} + \beta_j p_j$.

We can obtain the Lanczos vectors q_k and matrices T_k with (see [**?**])

$$\begin{aligned} q_k &= g_k / \|g_k\| \\ T_k &= \begin{bmatrix} \frac{1}{\alpha_0} & -\frac{\sqrt{\beta_0}}{\alpha_0} & & & & \\ -\frac{\sqrt{\beta_0}}{\alpha_0} & \frac{1}{\alpha_1} + \frac{\beta_0}{\alpha_0} & -\frac{\sqrt{\beta_1}}{\alpha_1} & & & \\ & -\frac{\sqrt{\beta_1}}{\alpha_1} & \frac{1}{\alpha_2} + \frac{\beta_1}{\alpha_1} & & & \\ & & & \ldots & & \\ & & & & \frac{1}{\alpha_{k-1}} + \frac{\beta_{k-2}}{\alpha_{k-2}} & -\frac{\sqrt{\beta_{k-1}}}{\alpha_{k-1}} \\ & & & & -\frac{\sqrt{\beta_{k-1}}}{\alpha_{k-1}} & \frac{1}{\alpha_k} + \frac{\beta_{k-1}}{\alpha_{k-1}} \end{bmatrix}. \end{aligned}$$

In addition, another advantage of the preconditioned conjugate gradient method is that if $q(x)$ is convex in the space $\mathcal{K}_{k+1}$, the minimizer of the next iteration can be easily calculated by:

$$x_{k+1} = x_k + \alpha_k p_k,$$

where $x_0 = 0$ and x_k is the minimizer of the k-th iteration. Finally, the vector g_{k+1}, obtained as a part of the computation process, is exactly the gradient of $q(x)$ at the point $x = x_{k+1}$. Therefore, if $g_{k+1} = 0$ or its value is sufficiently small, we know that it is already optimal and the algorithm terminates.

Having all the previous results in hand, we are now ready to describe the algorithm. In each iteration, we perform the preconditioned conjugate gradient method and use the results to obtain the Lanczos vectors and matrices. Then we solve the following problem:

$$\begin{array}{rll} h_k = & \operatorname{argmin} & h^T T_k h - 2\gamma_0 e_1^T h \\ & s.t. & \|h\| \leq s. \end{array} \tag{7}$$

Notice that this is a problem in k-dimension, where k is usually much smaller than n. Moreover, since the matrix T_k is tridiagonal, we know that (**??**) can be easily solved by many other algorithms, where the special structure of the matrix can also be exploited. For instance, we can simply use the MS algorithm to solve the problem, in which the Cholesky factorization is the main cost. But when the matrix is tridiagonal, the Cholesky factorization becomes much faster, making the algorithm much more efficient.

Once (**??**) is solved, we see that the minimizer of the problem:

$$\begin{array}{ll} \min & x^T A x - 2a^T x \\ s.t. & \|x\| \leq s \\ & x \in \mathcal{K}_k \end{array}$$

can be obtained by

$$x_k = Q_k h_k.$$

Since x_k is the solution to the TRS in the space $\mathcal{K}_k$, we may consider it to be an approximate solution to the original problem. However, we also need to know how good this approximation is, which means the optimality condition equation $(A - \lambda_k I)x_k = a$ has to be checked. This can be done by the following equality:

$$\|(A - \lambda_k I)x_k - a\| = \gamma_{k+1}|e_{k+1}^T h_k|.$$

Therefore, when this value is sufficiently small, the algorithm terminates and x_k at the current iteration is the solution we are looking for. The algorithm also gives the error of the solution, which allows the user to obtain a solution with desired accuracy.

However, this algorithm cannot handle the hard case, even though the authors in [**?**] present several results showing how to detect it.

3.3 The SSM Algorithm

Hager [**?**] developed a similar method to GLTR, which is called the sequential subspace method (SSM). There are two phases in this algorithm, where the first phase is very similar to the GLTR algorithm, and it is only used to generate an initial guess. In the second phase, the problem is solved over a subspace which is adjusted in successive iterations. However, instead of the Krylov subspace, the algorithm constructs a subspace with dimension 4, so that in each iteration the problem is easy to solve and global convergence is also guaranteed.

In each iteration, suppose (x_k, λ_k) is the current estimate we have, such that $\|x_k\| = s$ is satisfied. We first use the *sequential quadratic programming* to obtain a descent direction.

$$\begin{array}{ll} \min & q_{SQP}(z) = (x_k + z)^T A(x_k + z) - 2a^T(x_k + z) \\ s.t. & x_k^T z = 0. \end{array} \tag{8}$$

It can be proved that this optimization problem is equivalent to the following system of equations:

$$
\begin{aligned}
x_{SQP} &= x_k + z && \text{(9a)}\\
\lambda_{SQP} &= \lambda_k + \nu && \text{(9b)}\\
(A - \lambda_k I)z + \nu x_k &= a - (A - \lambda_k I)x_k && \text{(9c)}\\
x_k^T z &= 0. && \text{(9d)}
\end{aligned}
$$

Then, we move to the main step of the iteration —— solving the TRS over a subspace S_k. i.e.,

$$
\begin{array}{ll}
\min & q(x) = x^T A x - 2a^T x,\\
s.t. & \|x\| \leq s\\
 & x \in S_k,
\end{array}
\tag{10}
$$

where $S_k = \text{span}\{x_{SQP}, x_k, a - Ax_k, v_{\min}\}$. x_{SQP} is obtained from (**??**), and $v_{\min}$ is the eigenvector corresponding to $\lambda_{\min}(A)$. There are several reasons for constructing S_k this way. x_{SQP} is the minimizer to (**??**), and so it is an approximate solution to the TRS. By including x_k, it guarantees that the value of the objective function can only decrease in each iteration. $a - Ax_k$, which is the gradient of the objective function, is one of the steepest descent directions when the first-order optimality condition is not satisfied. The eigenvector corresponding to the smallest eigenvalue dislodges the iterates from a non-optimal stationary point. We also need to include this vector to keep $A - \lambda_k I$ positive definite (or positive semi-definite).

Also, even though we can obtain the solution of the primal variable x_{k+1}, it is usually too expensive to calculate the optimal value of the dual variable λ_{k+1} directly. Therefore, in order to obtain an estimate for the dual variable λ_{k+1}, we choose it such that the residual of the optimality conditions is minimized. i.e.,

$$
\lambda_{k+1} := \quad \text{argmin} \quad g(\lambda) = \|(A - \lambda I)x_{k+1} - a\|. \tag{11}
$$

This algorithm is shown to be effective when the easy case holds and the problem is non-degenerate (i.e., λ^* is much less than $\lambda_{\min}(A)$). When the hard case or almost hard case occurs, some modifications are also introduced to improve the effectiveness of the algorithm.

3.4 The Gould-Robinson-Thorne Algorithm

The Gould-Robinson-Thorne (GRT) algorithm [**?**] is building on the work of the MS algorithm, aiming at finding the root of the secular equation with fewer iterations. This improvement is achieved by using high-order polynomial approximation to the secular equation. This method is proved to be both globally and asymptotically super-linearly convergent.

Since our goal is to find a pair of x^* and λ^* satisfying the optimality conditions, recall that $(A - \lambda I)x = a$ is always consistent and we define:

$$
x(\lambda) = \quad \begin{array}{cc} \text{argmin} & \|x\| \\ \text{s.t.} & (A - \lambda I)x = a. \end{array} \tag{12}
$$

In addition, we define:

$$
\pi(\lambda, \beta) := \|x(\lambda)\|^{\beta}.
$$

Therefore, in order to obtain the optimal solution, we need to solve the equation:

$$\pi(\lambda, \beta) = s^{\beta} \tag{13}$$

for any $\beta \neq 0$. For example, $\pi(\lambda, -1)$ is indeed the secular equation in the MS algorithm, which is used to expedite Newton's method. Actually, it has also been shown that when $\beta = -1$, Newton's method gives the best approximation to the equation. The main improvement of this algorithm is that if we let $\beta = 2$ and define:

$$\pi(\lambda) := \|x(\lambda)\|^2,$$

then how the derivatives of $\pi(\lambda)$ can be calculated recursively is shown by the following theorem.

Theorem 3.2. *Suppose $(A - \lambda I)$ is positive definite, then for $k = 0, 1, ...,$ the derivatives of $\pi(\lambda)$ satisfy*

$$\begin{aligned}\pi^{(2k+1)}(\lambda) &= 2\alpha_k (x^{(k)})^T(\lambda)(x^{(k+1)})(\lambda) \\ \pi^{(2k+2)}(\lambda) &= \alpha_{k+1} (x^{(k+1)})^T(\lambda)(x^{(k+1)})(\lambda)\end{aligned}$$

where

$$\begin{aligned}(A - \lambda I)x^{(k+1)}(\lambda) &= -(k+1)x^{(k)}(\lambda) \\ \alpha_{k+1} &= \frac{2(2k+3)}{(k+1)}\alpha_k.\end{aligned}$$

■

Therefore, after knowing the derivatives of $\pi(\lambda)$, we can use the Taylor series to approximate the function, which gives a better estimated solution to the equation than Newton's method. Moreover, the process can be further controlled by the following result:

Theorem 3.3. *Suppose $\lambda < \min(\lambda_{\min}(A), 0)$, and let*

$$\pi_k(\delta) = \pi(\lambda) + \pi^{(1)}(\lambda)(\delta) + ... + \frac{1}{k!}\pi^{(k)}(\lambda)(\delta^k)$$

be the k-th order Taylor series approximation to $\pi(\lambda + \delta)$.
Then, when $\delta > 0$,

$$\pi(\lambda + \delta) \leq \pi_k(\delta), \text{ when } k \text{ is even},$$

$$\pi(\lambda + \delta) \geq \pi_k(\delta), \text{ when } k \text{ is odd}.$$

When $\delta < 0$,

$$\pi(\lambda + \delta) \geq \pi_{k+1}(\delta) \geq \pi_k(\delta), \quad \text{for all } k.$$

■

Therefore, in each iteration, an estimated solution as well as the upper bound and lower bound of the solution can be obtained by finding the root of the Taylor series, i.e., a high-order polynomial equation. The paper claims that by doing this, it takes much less iterations (usually 2 to 3) to obtain an accurate solution than the classic MS algorithm. The hard case can be handled similarly as the MS algorithm.

3.5 The RW Algorithm

To understand the RW Algorithm [?], we first consider a slightly different problem, which we denote as $TRS_=$:

$$(TRS_=) \qquad \begin{array}{rll} q^* = & \min & q(x) = x^T Ax - 2a^T x \\ & \text{s.t.} & \|x\| = s. \end{array}$$

The relation between TRS and $TRS_=$ is illustrated by the following theorem.

Theorem 3.4 ([?])**.** *Suppose $\bar{x}$ is the solution to the $TRS_=$ with the optimal dual variable λ^*. If $\lambda^* < 0$, then $\bar{x}$ is the solution to $\min(q(x) : \|x\| \leq \|s\|)$. If $\lambda^* \geq 0$, then $\bar{x}$ is the solution to $\min(q(x) : \|x\| \geq \|s\|)$.* ■

Therefore, if we can solve $TRS_=$ with a negative optimal dual variable, then we know it's also the solution to the TRS. On the other hand, if the optimal dual variable is not negative, we know that the unconstrained minimizer of $q(x)$ is inside the trust region (otherwise, $\bar{x}$ can not be the solution to $\min(q(x) : \|x\| \geq \|\bar{x}\|)$). In such case, we can obtain $x^* = A^{-1}a$ by conjugate gradient method.

To solve the $TRS_=$, we start by homogenizing the problem. Since we already know that strong duality holds for this problem (see [?]), we can obtain that:

$$\begin{aligned} q^* &= \min_{\|x\|=s} x^T Ax - 2a^T x \\ &= \min_{\|x\|=s,\eta^2=1} x^T Ax - 2\eta a^T x \\ &= \max_t \min_{\|x\|=s,\eta^2=1} x^T Ax - 2\eta a^T x + t(\eta^2 - 1) \\ &\geq \max_t \min_{\|x\|^2+\eta^2=s^2+1} x^T Ax - 2\eta a^T x + t(\eta^2 - 1) \\ &\geq \max_{t,\lambda} \min_{x,\eta} x^T Ax - 2\eta a^T x + t(\eta^2 - 1) + \lambda(\|x\|^2 + \eta^2 - s^2 - 1) \\ &= \max_{r,\lambda} \min_{x,\eta} x^T Ax - 2\eta a^T x + r(\eta^2 - 1) + \lambda(\|x\|^2 - s^2) \\ &= \max_{\lambda} (\max_r \min_{x,\eta} x^T Ax - 2\eta a^T x + r(\eta^2 - 1) + \lambda(\|x\|^2 - s^2)) \\ &= \max_{\lambda} \min_{x,\eta^2=1} x^T Ax - 2\eta a^T x + \lambda(\|x\|^2 - s^2) \\ &= \max_{\lambda} \min_x L(x, \lambda) \\ &= q^*, \end{aligned} \tag{14}$$

where $r = t + \lambda$, and $L(x, \lambda) = x^T Ax - 2a^T x - \lambda(\|x\|^2 - s^2)$ is the Lagrangian of the $TRS_=$.

Now we know all of the above expressions are equal, and so

$$q^* = \max_t \min_{\|x\|^2+\eta^2=s^2+1} x^T Ax - 2\eta a^T x + t(\eta^2 - 1).$$

If we define

$$z := \begin{pmatrix} \eta \\ x \end{pmatrix}, \quad D(t) := \begin{pmatrix} t & -a^T \\ -a & A \end{pmatrix}$$

and

$$k(t) := (s^2 + 1)\lambda_{\min}(D(t)) - t, \tag{15}$$

then

$$q^* = \max_t \min_{\|z\|^2=s^2+1} z^T D(t) z - t = \max_t k(t).$$

Therefore, the TRS$_=$ is equivalent to the unconstrained optimization problem $\max_t k(t)$. To solve this problem efficiently, we need to study more properties of the function:

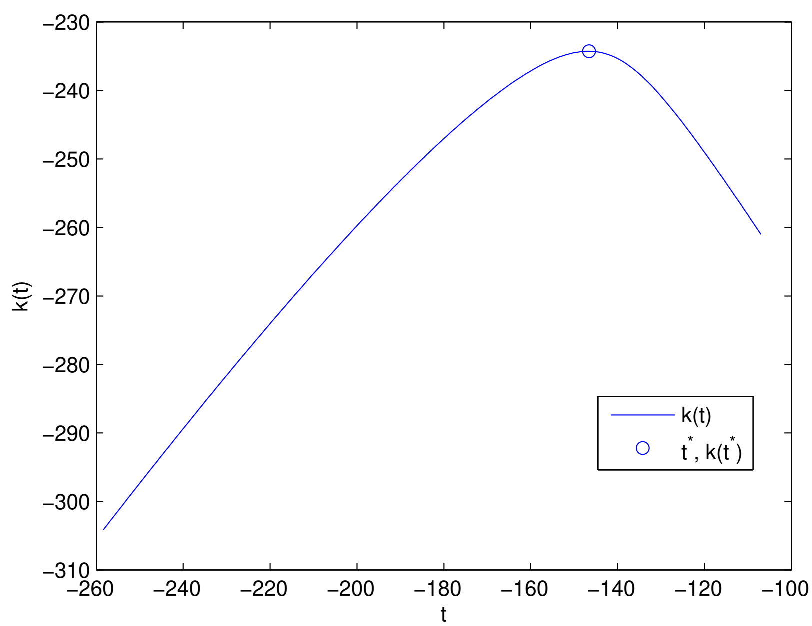
$$k(t) = (s^2+1)\lambda_{\min}(D(t)) - t.$$

It can be shown that $\lambda_{\min}(D(t))$ is a concave function for t (see [**?**]), and $k(t)$ is also a concave function (see Figure **??** and Figure **??**). It can also be shown that $\lim\limits_{t\to\infty} \lambda_{\min}(D(t)) = \lambda_{\min}(A)$, $\lim\limits_{t\to-\infty} \lambda_{\min}(D(t)) = t$. Therefore, the asymptotic behavior of $k(t)$ can be illustrated as follows:

$$\begin{array}{ll} k(t) \approx (s^2+1)\lambda_{\min}(A) - t, & \text{when } t \to \infty \\ k(t) \approx s^2 t, & \text{when } t \to -\infty. \end{array}$$

In addition, the properties of the derivative of $k(t)$ is also the key to maximizing it. If we let

Figure 5: k(t) in Easy Case



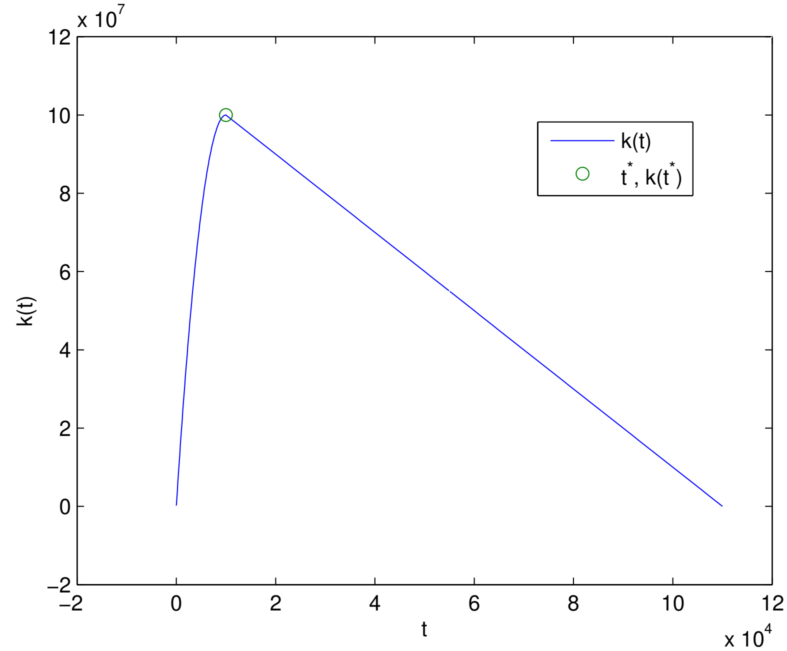
$y(t) := \begin{pmatrix} y_0(t) \\ w(t) \end{pmatrix}$ be the normalized eigenvector for $\lambda_{\min}(D(t))$, where $y_0(t) \in \mathbb{R}$ is the first component. It can be shown that (see [**?**]):

$$k'(t) = (s^2+1) y_0(t)^2 - 1.$$

When $k(t)$ is differentiable, if $t^* = \arg\max_t k(t)$, then

$$k'(t^*) = 0,$$

Figure 6: k(t) in Hard Case



$$y_0(t^*) = \frac{1}{\sqrt{s^2+1}}.$$

However, we may lose differentiability if $\lambda_{\min}(D(t))$ has more than one normalized eigenvector, i.e., its multiplicity is greater than 1. The following theorem shows the properties of the eigenvectors for $\lambda_{\min}(D(t))$ (see [?]):

Theorem 3.5. *Let $A = Q\Lambda Q^T$ be the spectral decomposition of A. Let $\lambda_{\min}(A)$ has multiplicity i and define*

$$t_0 := \lambda_{\min}(A) + \sum_{k \in \{j | (Q^T a)_j \neq 0\}} \frac{(Q^T a)_k^2}{\lambda_k(A) - \lambda_{\min}(A)}.$$

Then:
1. In the easy case, $\lambda_{\min}(D(t)) < \lambda_{\min}(A)$ and $\lambda_{\min}(D(t))$ has multiplicity 1.
2. In the hard case,
(a) for $t < t_0$, $\lambda_{\min}(D(t)) < \lambda_{\min}(A)$ and $\lambda_{\min}(D(t))$ has multiplicity 1,
(b) for $t = t_0$, $\lambda_{\min}(D(t)) = \lambda_{\min}(A)$ and $\lambda_{\min}(D(t))$ has multiplicity i+1,
(c) for $t > t_0$, $\lambda_{\min}(D(t)) = \lambda_{\min}(A)$ and $\lambda_{\min}(D(t))$ has multiplicity i.

■

Moreover, we have the following theorem showing how x^* and λ^* can be calculated from t^*.

Theorem 3.6. *Let $t^* = \operatorname{argmax}_t k(t)$, $y(t^*) = \begin{pmatrix} y_0(t^*) \\ w(t^*) \end{pmatrix}$ be the normalized eigenvector for $\lambda_{\min}(D(t^*))$, where $y_0(t^*)$ is the first component.*

If $\lambda_{\min}(D(t^*)) < 0$, then

$$x^* = \frac{w(t^*)}{y_0(t^*)}$$

$$\lambda^* = \lambda_{\min}(D(t^*))$$

is the solution to the (TRS).

If $\lambda_{\min}(D(t^*)) \geq 0$, then the minimizer is in the interior of the trust region, and so

$$x^* = A^{-1}a$$

$$\lambda^* = 0$$

is the solution to the (TRS).

■

3.5.1 Techniques Used for Maximizing $k(t)$

We define the range of "good side" and "bad side" for the variable t as follows:
If $k'(t) < 0$ $(t > t^*)$, then we say t is on the good side.
If $k'(t) > 0$ $(t < t^*)$, then we say t is on the bad side.
The reason why it is defined like this is quite simple: if we can find a t such that $t < t_0$ and $t > t^*$ (on the good side), then the hard case does not occur.

Bracketing Newton's Method

Since the second derivative of $k(t)$ involves the second derivative of the smallest eigenvalue in a large-scale parametric matrix, it is too expensive to calculate. Therefore, the classic Newton's method, which aims at finding the root of $k'(t)$, is not feasible in our situation. But the bracketing Newton's method ([**?**]) can be used to maximize $k(t)$.

Suppose we already have an upper bound and a lower bound of $k^* = \max_t k(t)$, denoted by k_{up} and k_{low} respectively, and the value of t is t_j in the current iteration. Assume α is a constant in $(0,1)$. We define

$$M_j := \alpha k_{up} + (1-\alpha)k_{low}.$$

Then we use Newton's method to solve the equation $k(t) - M_j = 0$ for one iteration (see Figure **??**). i.e.,

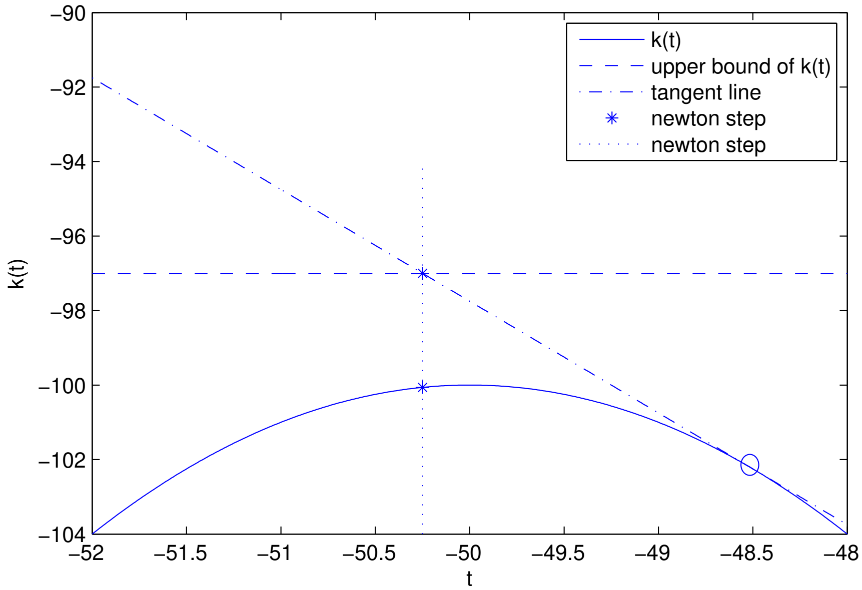
$$\begin{aligned} t_+ &= t_j - \tfrac{k(t_j)-M_j}{k'(t_j)} \\ &= \tfrac{(s^2+1)\lambda_{\min}(D(t_j))-M_j}{(s^2+1)y_0^2(t_j)-1} \end{aligned}$$

where t_+ is the estimate of the next iteration.

However, it is observed that $k(t_+) > k(t_j)$ is not guaranteed to be true, and so in some iterations, we may not have an improvement on the value of $k(t)$. But still, when this happens, it gives us some information about the bounds of k^*. Precisely, we define our steps as follows:
When $k(t_+) > k(t_j)$,

$$t_{j+1} = t_+$$

Figure 7: Bracketing Newton's Method



$$k_{low} = k(t_+).$$

When $k(t_+) \leq k(t_j)$,

$$t_{j+1} = t_j$$

$$k_{up} = M_j.$$

Global convergence and superlinear convergence is proved in [**?**], as long as $k(t)$ is a concave function and is bounded above, which is obviously true in our situation. However, the approximation given by this technique is not quite ideal, and therefore it is only used when other techniques fail.

Triangle Interpolation

Suppose in the past few iterations, we already have values of t on both the good side and the bad side, i.e., a t_g such that $k'(t_g) < 0$ and a t_b such that $k'(t_b) > 0$. We can find the secant lines on these two points, and set the intersection of them to be our new approximation t_+. Precisely, we solve the following linear equations:

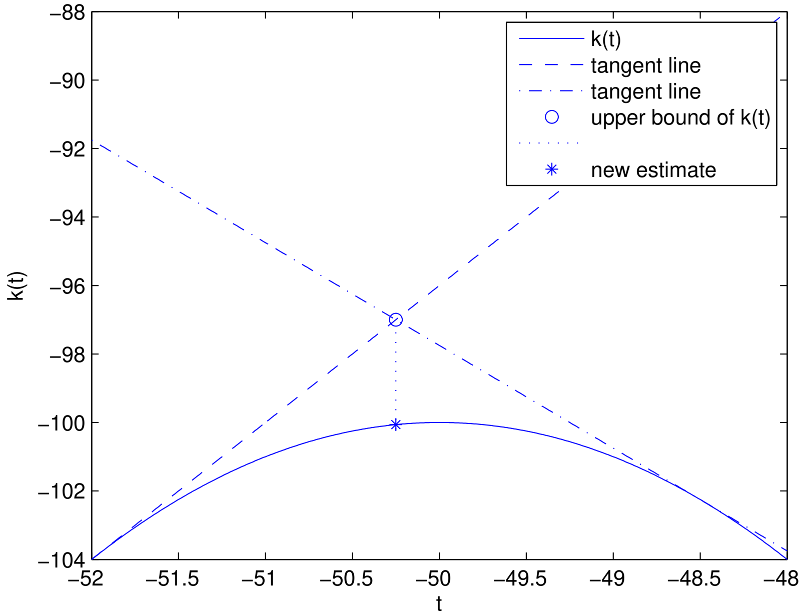
$$q - k(t_g) = k'(t_g)(t_+ - t_g)$$

$$q - k(t_b) = k'(t_b)(t_+ - t_b)$$

where q and t_+ are unknown variables. By doing this, if $t_+ \in (t_b, t_g)$, then not only t_+ is a new approximation, we can also conclude that q is an upper bound of $k(t)$, and therefore we have $q \geq k^*$ (see Figure **??**).

However, we should also be aware that t_+ is not always in (t_b, t_g). When $t_+ < t_b$ or $t_+ > t_g$, the triangle interpolation fails.

Figure 8: Triangle Interpolation



Vertical Cut

Suppose we have two values of t , a t_g such that $k'(t_g) < 0$ and a t_b such that $k'(t_b) > 0$, and further more $k(t_g) < k(t_b)$, then we can use vertical cut to reduce the interval of the t value. Concretely, we solve the equation

$$k(t_g) + k'(t_g)(t - t_g) = k(t_b).$$

This is the intersection of the secant line on $(t_g, k(t_g))$ and the line $k = k(t_b)$ (see Figure **??**). The value of t must be in (t_b, t_g) and it's also on the good side.

On the other hand, if $k(t_b) < k(t_g)$, we can also do it similarly by solving the equation

$$k(t_b) + k'(t_b)(t - t_b) = k(t_g).$$

Inverse Linear Interpolation

Since $k'(t) = (s^2 + 1)y_0(t)^2 - 1$, the following two equations are equivalent.

$$k'(t) = 0,$$
$$\psi(t) = \sqrt{s^2 + 1} - \frac{1}{y_0(t)}.$$

Therefore, we can use inverse linear interpolation on $\psi(t)$. We approximate the inverse function of $\psi(t)$ to be a linear function, i.e.,

$$t(\psi) = a\psi + b.$$

Figure 9: Vertical Cut

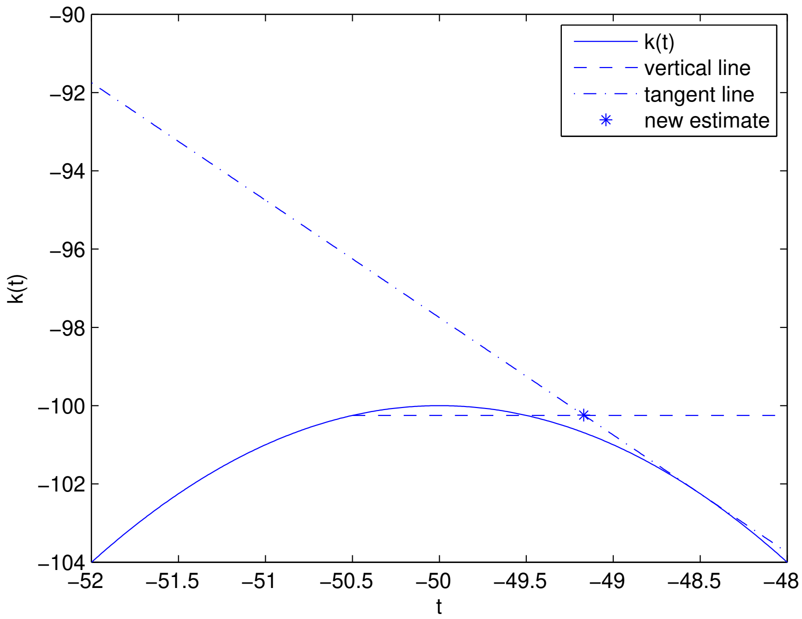
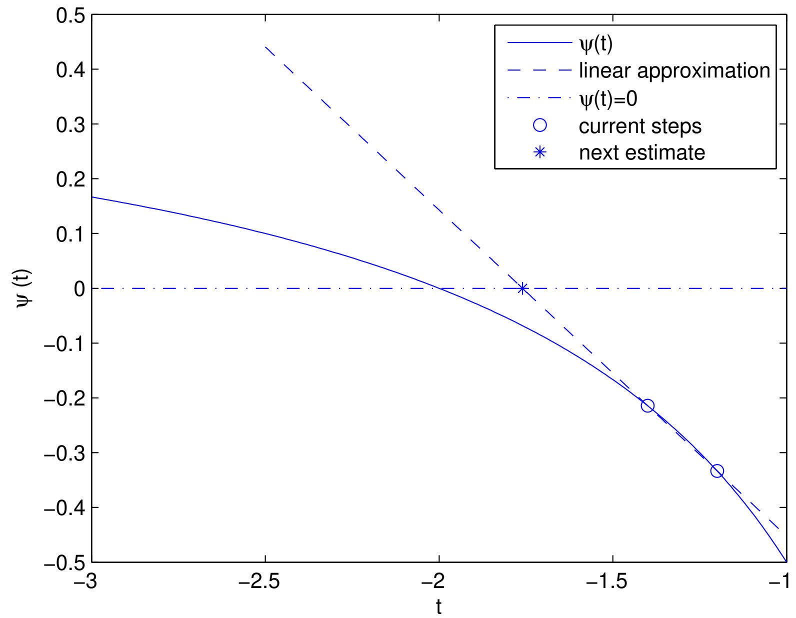


Figure 10: Linear Interpolation



So if two points (t_1, ψ_1) and (t_2, ψ_2) are given, we can calculate the coefficient of the linear inverse function and obtain a new estimate of t by $t(\psi) = t(0) = b$ (see Figure **??**).

Notice if the value of t of two points are on the same side (both on the good side or both on the bad side), the estimate must be on the good side. However, if they are on different sides, then the estimate must be on the bad side. Also, in the hard case, when $t > t_0$, $\psi(t)$ is not defined, and therefore if the value of t we obtained is greater than t_0, we can not use this technique. However, we do not know if the hard case occur or not, unless we have a point on the good side. Therefore, when we have no points on the good side, this is indeed an extrapolation, which may result in an irrelevant approximation. So we should not use this technique at all until we have at least one point on the good side.

3.5.2 Flow Chart

Algorithm 3.5. ***The RW Algorithm***

1. ***Initialize***
   Estimate $\lambda_{\min}(A)$, the upper bound and the lower bound of μ^* and t^*. (i.e., μ_{up}, μ_{low}, t_{up}, t_{low})

2. ***The Main Loop***
   While $(t_{up} - t_{low} > \mathrm{tol}_t)$ & $(\mu_{up} - \mu_{low} > \mathrm{tol}_\mu)$ & $(|k'(t)| > \mathrm{tol}_{dk})$ & (optimality condition is not satisfied)
   ***Take Steps to Maximize** $k(t)$*

   (a) *Update t by midpoint (i.e., $t = (t_{up} + t_{low})/2$).*

   (b) *Update t by bracketing Newton's method.*

   (c) *If we have points on both the good side and the bad side,*

      i. *update μ_{up} and t by triangle interpolation;*

      ii. *update t_{up} or t_{low} by vertical cut.*

   (d) *Update t by inverse linear interpolation.*

3. ***Updates***
   Calculate $\lambda_{\min}(D(t))$ and $y(t)$ with the new value of t.
   If $\lambda_{\min}(D(t)) > 0$, then the minimizer is in the interior of the trust region,
   calculate $x^* = A^{-1}a$ by preconditioned conjugate gradient method, then the algorithm terminates.
   Update the value of μ_{up}, μ_{low}, t_{up}, t_{low}, $k(t)$, $k'(t)$, etc.

Remark 3.1.

1. *In the steps taken to maximize $k(t)$, the value of t is updated only if the new estimated t_+ satisfies $t_{low} < t_+$ and $t_+ < t_{up}$. Otherwise, the value of t is not changed.*

2. *All the techniques are used to update the value of t if it's applicable. The order of the techniques used is (i) bracketing Newton's method; (ii) triangle interpolation; (iii) vertical cut; (iv) inverse linear interpolation. Since the latter techniques are considered to have better approximations than the former ones, the value of t can be overwritten by the latter techniques.*

3. *From the review of the existing algorithms, we can divide them into two types. One is trying to solve λ^* based on Cholesky factorization, while the other one is based on Krylov methods, where sparsity and structure can be exploited. Also, we can find that the hard case problems usually can not be solved by ordinary routines, and so most algorithms have special techniques to handle the hard case once it is detected.*

4 Simple Scaling for the TRS

4.1 Scaling

In this section, we show that although a theoretically equivalent TRS can be obtained by simple scaling, this process can have significant effect on the RW algorithm. We can take advantage of this fact to improve the algorithm and obtain better results. Moreover, scaling can also give rise to some different approaches to solve the TRS.

Suppose $r > 0$ is a scalar. We can obtain an equivalent trust region subproblem:

$$(TRS_r) \qquad \begin{array}{rll} \hat{q}^* = & \min & \hat{q}(x) := x^T \hat{A} x - 2\hat{a}^T x \\ & \text{s.t.} & \|x\| \leq \hat{s}, \end{array} \tag{16}$$

where

$$\hat{A} = r^2 A, \quad \hat{a} = ra, \quad \hat{s} = \frac{1}{r} s.$$

Lemma 4.1. *Let $S = \{x | q(x) = q^*, \|x\| \leq s\}$ be the solution set of (TRS), $\hat{S} = \{x | \hat{q}(x) = \hat{q}^*, \|x\| \leq \hat{s}\}$ be the solution set of (TRS_r), then*

$$q^* = \hat{q}^*, \quad \frac{1}{r} S = \hat{S},$$

where $\frac{1}{r}S = \{\frac{1}{r}x | x \in S\}$.

Proof: If $x^* \in S$ is an optimal solution to (TRS), then $q^* = q(x^*)$ and $\|x^*\| \leq s$. Therefore, $\|\frac{1}{r}x^*\| \leq \frac{1}{r}s = \hat{s}$, and so $\frac{1}{r}x^*$ is a feasible solution to (TRS_r). Then we have

$$q^* = q(x^*) = \hat{q}(\frac{1}{r}x^*) \geq \hat{q}^*. \tag{17}$$

Also, if $\hat{x}^* \in \hat{S}$ is an optimal solution to (TRS_r), then we know that $r\hat{x}^*$ is feasible for (TRS), and so

$$\hat{q}^* = \hat{q}(\hat{x}^*) = q(r\hat{x}^*) \geq q^*. \tag{18}$$

Combining (**??**) and (**??**), we have $q^* = \hat{q}^*$.
Moreover, $\forall x^* \in S$, $\|\frac{1}{r}x^*\| \leq \hat{s}$, and from (**??**) we know that $\hat{q}(\frac{1}{r}x^*) = q^* = \hat{q}^*$. Therefore, $\frac{1}{r}x^*$ is an optimal solution to (TRS_r), and so $\frac{1}{r}S \subseteq \hat{S}$. On the other hand, $\forall \hat{x}^* \in \hat{S}$, $\hat{x}^* = r(\frac{1}{r}\hat{x}^*)$, where $r\hat{x}^* \in S$. Therefore, $\exists x^* \in S$ such that $\hat{x}^* = \frac{1}{r}x^*$, and so we have $\frac{1}{r}S \supseteq \hat{S}$. The result $\frac{1}{r}S = \hat{S}$ follows.

■

We now consider r as a parameter in our algorithm; and, we let $R = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & rI \end{pmatrix}$, and define

$$\begin{aligned} D(r,t) &:= \begin{pmatrix} t & -ra^T \\ -ra & r^2 A \end{pmatrix} \\ &= RD(t)R. \end{aligned}$$

Then the function $k(t)$ in the RW algorithm can be viewed as a bivariate function:

$$\hat{k}(r,t) := \left(\frac{s^2}{r^2}+1\right)\lambda_{\min}(D(r,t)) - t, \forall r > 0.$$

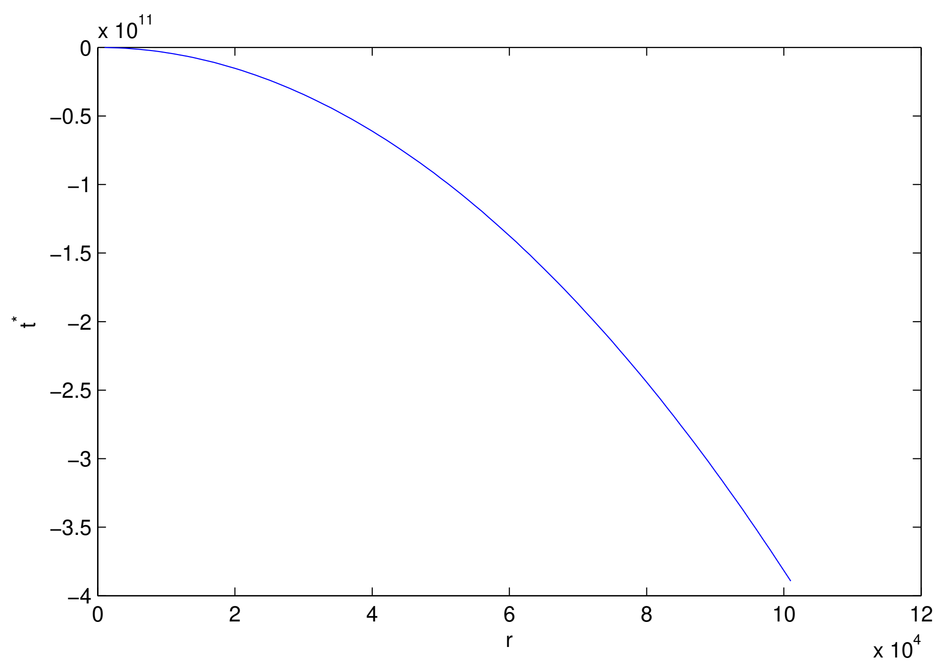
Since $\hat{k}(r,t)$ is a concave function with respect to t, we have $\max_t \hat{k}(r,t)$ is equivalent to solving $\frac{d\hat{k}}{dt} = 0$, when $\hat{k}(r,t)$ is differentiable. Moreover, observe that for any fixed $\bar{r} > 0$, we have

$$\max_{r>0,t} \hat{k}(r,t) = \max_t \hat{k}(\bar{r},t) = q^*.$$

Therefore, we know that $\forall r > 0$, there is a unique t^* (which depends on r) such that $\hat{k}(r^*, t(r^*)) = q^*$. As a result, the optimal value of t can be viewed as a function of r, i.e., $t^* = t(r)$ (see Figure **??**).

Conjecture 4.1. *$t^* = t(r)$ is a decreasing and concave function with respect to r.*

Figure 11: $t(r)$



4.2 How Scaling Affects Optimal t^*

1. The initialization of the upper bound and the lower bound of t in (TRS) is given by (see [?]):

$$t_{low} = \lambda_{\min}(A) - \frac{\|a\|}{s}, \quad t_{up} = \lambda_{\min}(A) + s\|a\|.$$

   Therefore, in the scaled problem (TRS_r), the upper bound and the lower bound become:

$$t_{low} = \lambda_{\min}(r^2 A) - r^2\frac{\|a\|}{s}, \quad t_{up} = \lambda_{\min}(r^2 A) + s\|a\|.$$

2. We can see that the initial interval of t is (see Figure **??**):

$$t_{up} - t_{low} = (s + \frac{r^2}{s})\|a\|. \tag{19}$$

   The RW algorithm keeps searching for the optimal value of t in this interval and updates the upper bound and the lower bound iteratively, until the interval is smaller than the tolerance or some other stopping criteria are met. As a result, under the same accuracy, the smaller the initial interval is, the faster the algorithm converges. Therefore, from (**??**) we know that we can decrease the initial interval of t and accelerate the algorithm by decreasing the value of r. As $r \to 0$, $(t_{up} - t_{low}) \to s\|a\|$.

3. Observe from the graph that when r increases, t^* is also moving towards t_{up} (see Figure **??**). In the first iteration of the algorithm, we don't have any points available, and the midpoint method is the only technique performed. Hence when t^* is close to t_{up}, it is "far away" from t_{low}, and so the first estimation of t (the midpoint of t_{up} and t_{low}) is on the bad side. Conversely, if r is small, the first estimation of t is on the good side (see Figure **??**). Based on this observation, the following three strategies may be used to improve the algorithm.

   (a) Set r to be large, and start searching for t^* from t_{up}.
   (b) Set r to be small, and start searching for t^* from t_{low}.
   (c) Find a "good" estimation of r so that t^* is almost the midpoint of t_{up} and t_{low}.

4.3 How to Choose the Best Scalar

How the scalar affects the algorithm is complicated, but we can still have some other choice of r which is motivated by accelerating the convergence rate for maximizing $k(t)$. Recall that the asymptotic behavior of $k(t)$ is

$$\begin{array}{ll} k(t) \approx (s^2+1)\lambda_{\min}(A) - t, & \text{when } t \to \infty \\ k(t) \approx s^2 t, & \text{when } t \to -\infty. \end{array}$$

Therefore, we can have the asymptotic behavior of $k'(t)$

$$\begin{array}{ll} k'(t) \approx -1, & \text{when } t \to \infty \\ k'(t) \approx s^2, & \text{when } t \to -\infty. \end{array}$$

Figure 12: Upper Bound and Lower Bound of t When $r = 1$ (Not Scaled)

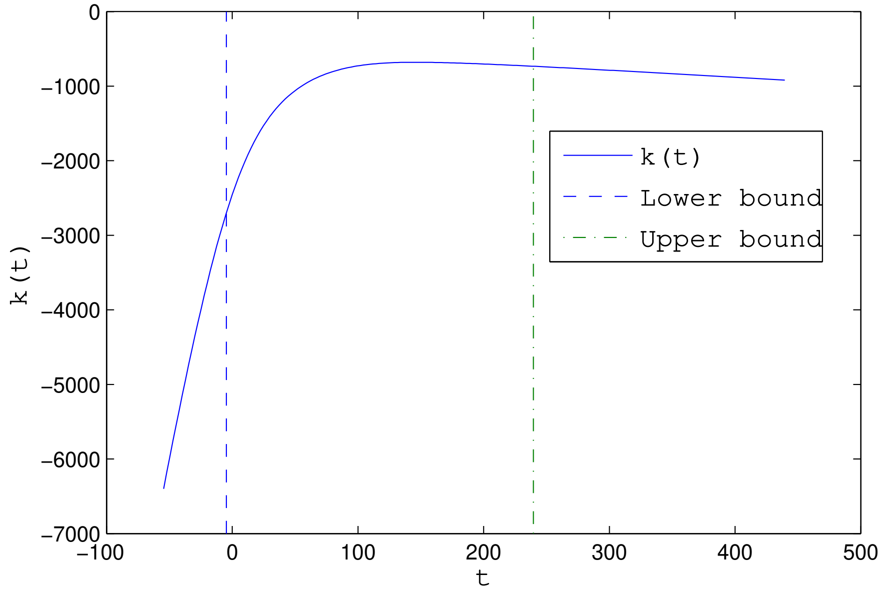


Figure 13: Upper Bound and Lower Bound of t When $r = 1000$

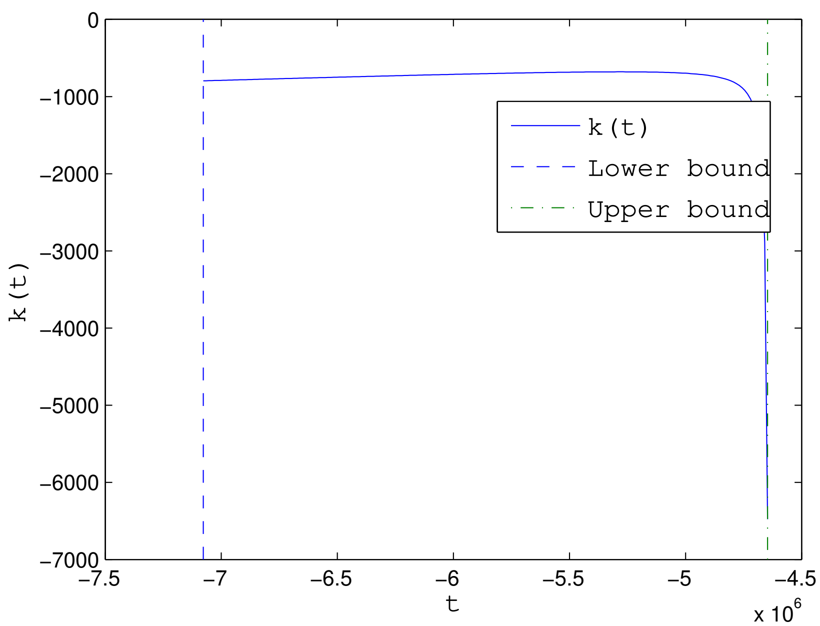
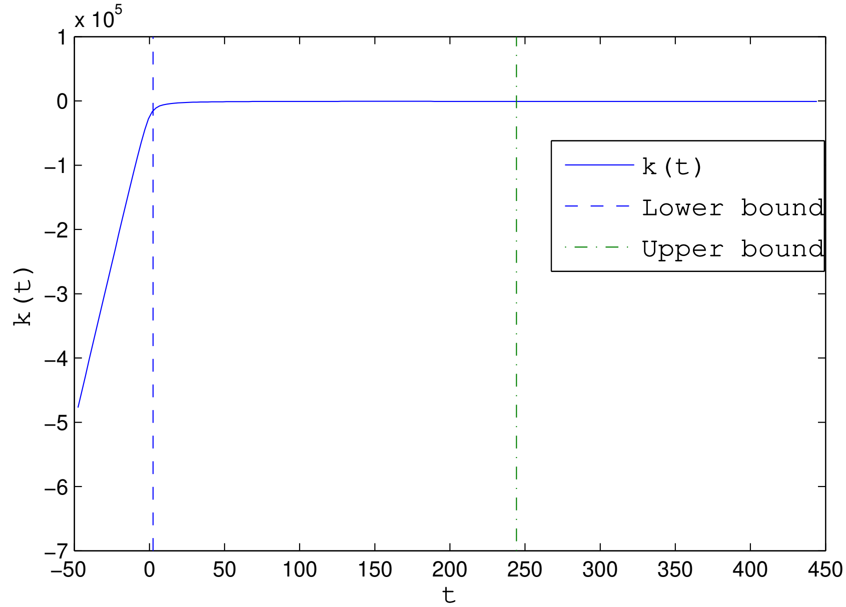


Figure 14: Upper Bound and Lower Bound of t When $r = 1/1000$



If we set $r = s$, the radius of the scaled problem becomes $s/r = 1$, and so the shape of $k(t)$ is more symmetric because $k'(t) \approx 1$ for t sufficiently small and $k'(t) \approx -1$ for t sufficiently large. This can significantly improve the performance of the triangle interpolation for maximizing $k(t)$. Numerical results are given in Section **??** to provide the performance of this choice of scalar.

4.4 The Gradient and Hessian of $\hat{k}(r,t)$

As we have mentioned above, instead of fixing the value of r, we can view r as a variable of k, and maximize $\hat{k}(r,t)$ by changing the value of both t and r simultaneously in each iteration. Therefore, unconstrained optimization methods like steepest descent can be applied to maximize $\hat{k}(r,t)$. However, we should examine more properties of the function $\hat{k}(r,t)$ first.

Theorem 4.1. *Let*

$$\hat{k}(r,t) = (\frac{s^2}{r^2} + 1)\lambda_{\min}(D(r,t)) - t,$$

where

$$D(r,t) \quad = \begin{pmatrix} t & -ra^T \\ -ra & r^2 A \end{pmatrix}.$$

Let $A = Q\Lambda Q^T$ be the spectral decomposition of A, γ_k be the kth component of $Q^T a$, and $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots \lambda_n)$ such that $\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$ be the eigenvalues of A. Then,

1. *In the easy case, $\hat{k}(r,t)$ is differentiable on $(0, +\infty) \times \mathbb{R}$.*

2. In the hard case, let

$$t_0 = r^2\lambda_1 + \sum_{k\in\{j|\gamma_j\neq 0\}} \frac{\gamma_k^2}{\lambda_k - \lambda_1}.$$

Then $\hat{k}(r,t)$ is differentiable on $(0,+\infty)\times(-\infty,t_0)$.

The gradient of $\hat{k}(r,t)$ is given by

$$\nabla\hat{k}(r,t) = \begin{pmatrix} \frac{\partial \hat{k}(r,t)}{\partial r} \\ \frac{\partial \hat{k}(r,t)}{\partial t} \end{pmatrix} = \begin{pmatrix} -2s^2r^{-3}\lambda_{\min}(D(r,t)) - 2rw(r,t)^TAw(r,t) + 2y_0(r,t)a^Tw(r,t) \\ (s^2r^{-2}+1)y_0^2(r,t) - 1 \end{pmatrix}$$

where $y(r,t) = \begin{pmatrix} y_0(r,t) \\ w(r,t) \end{pmatrix}$ is the normalized eigenvector corresponding to $\lambda_{\min}(D(r,t))$.

Moreover, if A has n distinct eigenvalues, i.e., $\lambda_1(A) < \lambda_2(A) < \ldots < \lambda_n(A)$, and for $j = 1,...,n$, $\gamma_j \neq 0$, then $\hat{k}(r,t)$ is twice differentiable on $(0,+\infty)\times\mathbb{R}$. The Hessian of $\hat{k}(r,t)$, a 2×2 symmetric matrix H, is given by:

$$H(1,1) = \frac{\partial^2\hat{k}(r,t)}{\partial r^2} = (6s^2r^{-4})\lambda_{\min}(D) - 4s^2r^{-3}(y^TD_ry) + (s^2r^{-2}+1)(2w^TAw + 8\sum_i \frac{(rw^TAw - y_0a^Tw)^2}{\lambda_{\min}(D)-\lambda_i(D)})$$

$$H(1,2) = H(2,1) = \frac{\partial^2\hat{k}(r,t)}{\partial r\partial t} = -2s^2r^{-3}y_0^2 + 2(s^2r^{-2}+1)(\sum_i y_0y_0^i \frac{2rw^TAw^i - a^T(y_0^iw + y_0w^i)}{\lambda_{\min}(D)-\lambda_i(D)})$$

$$H(2,2) = \frac{\partial^2\hat{k}(r,t)}{\partial t^2} = 2(s^2r^{-2}+1)(\sum_i \frac{(y_0y_0^i)^2}{\lambda_{\min}(D)-\lambda_i(D)})$$

where $y^i = \begin{pmatrix} y_0^i \\ w^i \end{pmatrix}$ is the normalized eigenvector corresponding to the eigenvalue $\lambda_i(D)$, such that $\lambda_i(D) \neq \lambda_{\min}(D)$.

Proof:
Without loss of generality, we assume A is diagonal, and so $A = \Lambda$, $Q = Q^T = I$ and $Q^Ta = a$. From [**?**], we have

$$\det(D(r,t) - \lambda I) = (t-\lambda)\prod_{k=1}^n (r^2\lambda_k - \lambda) - \sum_{k=1}^n((ra_k)^2 \prod_{j\neq k}^n (r^2\lambda_j - \lambda)).$$

Therefore, if we let

$$d(\lambda) = \lambda + \sum_{k=1}^n \frac{(ra_k)^2}{r^2\lambda_k - \lambda},$$

when $\lambda \neq r^2\lambda_k$, we have

$$\det(D(r,t) - \lambda) = (t - d(\lambda))\prod_{k=1}^n (r^2\lambda_k - \lambda). \tag{20}$$

In the easy case, we know that at least one component of a corresponding to the smallest eigenvalue is nonzero. Without loss of generality, assume $a_1 \neq 0$, so we have

$$\lim_{\lambda \to -\infty} d(\lambda) = -\infty,$$

$$\lim_{\lambda \to (r^2\lambda_1)^-} d(\lambda) = \infty.$$

Moreover,

$$d'(\lambda) = 1 + \sum_{k=1}^{n} \frac{(ra_k)^2}{(r^2\lambda_k - \lambda)^2} > 0,$$

$$d''(\lambda) = \sum_{k=1}^{n} \frac{2(ra_k)^2}{(r^2\lambda_k - \lambda)^3}.$$

When $\lambda \in (-\infty, r^2\lambda_1)$, $d''(\lambda) > 0$ and $d(\lambda)$ is a convex and strictly increasing function, and its range is $(-\infty, \infty)$. Therefore, there is a unique solution $\lambda \in (-\infty, r^2\lambda_1)$ with multiplicity 1 for $t - d(\lambda) = 0$ for any fixed value of t. This solution must be the smallest eigenvalue of $D(r,t)$ (because it is the smallest solution for $\det(D(r,t) - \lambda I) = 0$). So we know that $\lambda_{\min}(D(r,t))$ has multiplicity 1. From [**?**], $\lambda_{\min}(D(r,t))$ is differentiable, and we know $\hat{k}(r,t)$ is also differentiable.
In the hard case, $a_k = 0$ for all $k \in \{j | \lambda_j = \lambda_1\}$. If $t < t_0$, then

$$\lim_{\lambda \to (r^2\lambda_1)^-} d(\lambda) = r^2\lambda_1 + \sum_{k \in \{j | \lambda_j > \lambda_1\}}^{n} \frac{(ra_k)^2}{r^2\lambda_k - r^2\lambda_1} = t_0 > t.$$

Since $d(\lambda)$ is still convex and strictly increasing when $\lambda \in (-\infty, r^2\lambda_1)$, there is still a unique solution with multiplicity 1 for $t - d(\lambda) = 0$ in $(-\infty, r^2\lambda_1)$. Therefore, $\hat{k}(r,t)$ is differentiable.
The gradient of $\hat{k}(r,t)$ can be calculated by

$$\begin{aligned}
\nabla \hat{k}(r,t) = \begin{pmatrix} \frac{\partial \hat{k}(r,t)}{\partial r} \\ \frac{\partial \hat{k}(r,t)}{\partial t} \end{pmatrix} &= \begin{pmatrix} -2s^2r^{-3}\lambda_{\min}(D(r,t)) - y(r,t)^T \frac{\partial D(r,t)}{\partial r} y(r,t) \\ (s^2r^{-2}+1)y_0^2(r,t) - 1 \end{pmatrix} \\
&= \begin{pmatrix} -2s^2r^{-3}\lambda_{\min}(D(r,t)) - y(r,t)^T \begin{pmatrix} 0 & -a^T \\ -a & 2rA \end{pmatrix} y(r,t) \\ (s^2r^{-2}+1)y_0^2(r,t) - 1 \end{pmatrix} \\
&= \begin{pmatrix} -2s^2r^{-3}\lambda_{\min}(D(r,t)) - 2rw(r,t)^T A w(r,t) + 2y_0(r,t)a^T w(r,t) \\ (s^2r^{-2}+1)y_0^2(r,t) - 1 \end{pmatrix}.
\end{aligned}$$

Moreover, if $\lambda_1 < \lambda_2 < \ldots < \lambda_n$, and every component of a is nonzero, then for $k = 1, 2, \ldots n-1$,

$$\lim_{\lambda \to (r^2\lambda_k)^+} d(\lambda) = -\infty,$$

$$\lim_{\lambda \to (r^2\lambda_k)^-} d(\lambda) = \infty.$$

Therefore, there is a unique solution $\lambda \in (r^2\lambda_k, r^2\lambda_{k+1})$ for $t - d(\lambda) = 0$. From (**??**), we conclude that this is an eigenvalue of $D(r,t)$. Also, since

$$\lim_{\lambda \to -\infty} d(\lambda) = -\infty,$$

$$\lim_{\lambda \to +\infty} d(\lambda) = \infty,$$

there are also unique solutions in the interval $(-\infty, r^2\lambda_1)$ and $(r^2\lambda_n, +\infty)$ respectively for $t - d(\lambda) = 0$. Therefore,

$$\lambda_1(D(r,t)) < r^2\lambda_1 < \lambda_2(D(r,t)) < r^2\lambda_2 < \ldots < \lambda_n(D(r,t)) < r^2\lambda_n < \lambda_{n+1}(D(r,t)).$$

By [**?**], if all the eigenvalues of $D(r,t)$ are distinct, the eigenvalues of $D(r,t)$ are all twice differentiable with respect to r and t, including $\lambda_{\min}(D(r,t))$. Hence $\hat{k}(r,t)$ is twice differentiable on $(0, +\infty) \times \mathbb{R}$.
Let

$$D_r = \frac{\partial D(r,t)}{\partial r} = \begin{pmatrix} 0 & -a^T \\ -a & 2rA \end{pmatrix},$$

$$D_{rr} = \frac{\partial^2 D(r,t)}{\partial r^2} = \begin{pmatrix} 0 & 0 \\ 0 & 2A \end{pmatrix},$$

then we have

$$\begin{aligned} \frac{\partial^2 \hat{k}(r,t)}{\partial r^2} &= (6s^2r^{-4})\lambda_{\min}(D) - 4s^2r^{-3}(y^T D_r y) + (s^2r^{-2}+1)(y^T D_{rr} y + 2\textstyle\sum_i \frac{(y^T D_r y)^2}{\lambda_{\min}(D) - \lambda_i(D)}) \\ &= (6s^2r^{-4})\lambda_{\min}(D) - 4s^2r^{-3}(y^T D_r y) + (s^2r^{-2}+1)(2w^T A w + 8\textstyle\sum_i \frac{(rw^T A w - y_0 a^T w)^2}{\lambda_{\min}(D) - \lambda_i(D)}), \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 \hat{k}(r,t)}{\partial r \partial t} &= -2s^2r^{-3}y_0^2 + 2(s^2r^{-2}+1)(\textstyle\sum_i y_0 y_0^i \frac{y^T D_r y}{\lambda_{\min}(D) - \lambda_i(D)}) \\ &= -2s^2r^{-3}y_0^2 + 2(s^2r^{-2}+1)(\textstyle\sum_i y_0 y_0^i \frac{2rw^T A w^i - a^T(y_0^i w + y_0 w^i)}{\lambda_{\min}(D) - \lambda_i(D)}), \end{aligned}$$

$$\frac{\partial^2 \hat{k}(r,t)}{\partial t^2} = 2(s^2r^{-2}+1)(\textstyle\sum_i \frac{(y_0 y_0^i)^2}{\lambda_{\min}(D) - \lambda_i(D)}).$$

■

5 Shift and Deflation

The technique of shift is widely used in eigenvalue problems, linear systems and many other matrix analysis problems. It can also be exploited in our algorithm to handle the hard case (case 2) (see [**?**]). We know that the hard case (case 2) happens only when $\lambda^* = \lambda_{\min}(A) \leq 0$. Therefore, if we can make A positive definite, then $\lambda_{\min}(A) > 0$, and so the hard case (case 2) does not occur. The following lemma tells us how the shift and deflation can be applied to make A positive definite.

Lemma 5.1. *Let $A = \sum_{i=1}^{n} \lambda_i(A) v_i v_i^T = Q \Lambda Q^T$ be the spectral decomposition of A, with v_i orthonormal eigenvectors and $Q = (v_1, v_2, ...v_n)$ an orthogonal matrix. Let $\gamma_i = (Q^T a)_i$, which is the i-th component of $Q^T a$. Let*

$$S_1 = \{i : \gamma_i \neq 0, \lambda_i(A) > \lambda_{\min}(A)\}.$$

$$S_2 = \{i : \gamma_i = 0, \lambda_i(A) > \lambda_{\min}(A)\}.$$

$$S_3 = \{i : \gamma_i \neq 0, \lambda_i(A) = \lambda_{\min}(A)\}.$$

$$S_4 = \{i : \gamma_i = 0, \lambda_i(A) = \lambda_{\min}(A)\}.$$

For $k = 1, 2, 3, 4$, let $A_k = \sum_{i \in S_k} \lambda_i(A) v_i v_i^T$. Then,

1. *If $S_3 \neq \emptyset$ (easy case), then*
   (x^*, λ^*) solves the TRS iff
   (x^*, λ^*) solves the TRS when A is replaced by $A_1 + A_3$.

2. *If $S_3 = \emptyset$ (hard case), let $i_0 = 1 \in S_4$, then*
   (x^*, λ^*) solves the TRS iff
   (x^*, λ^*) solves the TRS when A is replaced by $A_1 + \lambda_{i_0}(A) v_{i_0} v_{i_0}^T$.

3. *Let $x(\lambda^*) = (A - \lambda^* I)^\dagger a$, then*
   (x^*, λ^*), where $x^* = x(\lambda^*) + z, z \in \mathcal{N}(A - \lambda^* I)$ and $\|x^*\| = s$ solves the TRS iff
   $(x(\lambda^*), \lambda^* - \lambda_{\min}(A))$ solves the TRS when A is replaced by $A - \lambda_{\min}(A) I$.

4. *If $\lambda_{\min}(A) \geq 0$, then*
   (x^*, λ^*) solves the TRS iff
   (x^*, λ^*) solves the TRS when A is replaced by $A + \sum_{i \in S_4} \alpha_i v_i v_i^T$, with $\alpha_i \geq 0$.

■

Therefore, after we calculate the smallest eigenvalue $\lambda_{\min}(A)$ and the corresponding eigenvector v, we check whether $\lambda_{\min}(A) < 0$ and whether $v^T a = 0$ (or $v^T a$ is sufficiently small). If they are both true, we know that we may have the hard case. Then we replace A by $A - \lambda_{\min}(A) I$ (shift), and then deflate A by adding $\alpha_i v v^T$ to A. Now, if the smallest eigenvalue of A after the shift and deflation is positive, we know v is the only eigenvector corresponding to $\lambda_{\min}(A)$. But if the smallest eigenvalue of A after the shift and deflation is still 0 (or sufficiently small), then we know the multiplicity of the smallest eigenvalue is greater than 1, and so we need to find all the eigenvectors of $\lambda_{\min}(A)$. This is done by deflating all the eigenvectors we have found corresponding to the smallest eigenvalue iteratively (i.e., replace A by $A + \alpha_i v_i v_i^T$ if $A v_i = \lambda_{\min}(A) v_i$). If $v_i^T a \neq 0$ for some i such that $\lambda_i(A) = \lambda_{\min}(A)$, we know that we do not have the hard case, and we continue with the regular algorithm.

However, if $v_i^T a = 0$ for all i such that $\lambda_i(A) = \lambda_{\min}(A)$, then the hard case holds. Since we have deflated all the eigenvectors corresponding to $\lambda_{\min}(A)$ (notice that $\lambda_{\min}(A) = 0$ after the shift), A is now positive definite. Then we can calculate $x(\lambda^*) = A^{-1} a$, which is the solution to the shifted and deflated problem. If $\|x(\lambda^*)\| \leq s$, then we have the hard case (case 2). Therefore, $\lambda^* = \lambda_{\min}(A)$. According to Lemma **??** Item 3, $x^* = x(\lambda^*) + z$ is the optimal solution to the original

TRS with $z \in \mathcal{N}(A - \lambda^* I)$ and $\|x^*\| = s$. Recall that we have calculated at least one eigenvector v corresponding to $\lambda_{\min}(A)$, and we have $(A - \lambda_{\min}(A)I)v = Av - \lambda_{\min}(A)v = 0$. Therefore, $v \in \mathcal{N}(A - \lambda^* I)$, and we can set $z = v$ and scale it so that $\|x(\lambda^*) + z\| = s$. By doing this, we solve the TRS in the hard case (case 2), with the optimal solution $x^* = x(\lambda^*) + v$, $\lambda^* = \lambda_{\min}(A)$. If $\|x(\lambda^*)\| > s$, then we have the hard case (case 1), and this can be solved by the regular algorithm.

In summary, if the hard case (case 2) holds, the TRS can be solved with the cost of calculating the smallest eigenvalue of A and solving a well-conditioned linear systems. If $\lambda_{\min}(A)$ is not multiple, solving the hard case (case 2) is significantly faster than solving the easy case and the hard case (case 1), where we need to find the optimal value of t by the regular algorithm. More results will be shown in Section **??**.

Notice that if we find that we do not have the hard case (case 2) after the shift and deflation, we still keep the shift and deflation because it can still accelerate the algorithm. This is because after the shift and deflation, A becomes positive definite, and so λ^* is well separated from the smallest eigenvalue of A. Moreover, the solution can also be easily recovered by taking a primal step to the boundary.

Remark 5.1.

1. *If the shift and deflation is used (i.e., $\lambda_{\min}(A) < 0$ and $v^T a$ is sufficiently small), but we do not have the hard case (case 2), then we need to proceed with the regular algorithm. Notice that A becomes positive definite after the shift and deflation. Since the hard case (case 2) does not hold, for the original problem, we know that $\|x(\lambda_{\min}(A))\| > s$, and so the solution can not be in the interior for the shifted and deflated problem. According to Lemma **??** Item 3 and Item 4, x^* does not change after the shift and deflation. Therefore, we only need to recover the value of λ^* and q^* after we solve the shifted and deflated TRS.*

2. *The preconditioned conjugate gradient is applied right after we confirm that all the eigenvectors corresponding to the smallest eigenvalue are orthogonal to a (i.e., we have the hard case). If $x(\lambda^*)$ obtained from the preconditioned conjugate gradient is not in the trust region, we need to proceed with the regular algorithm, and this is somehow not efficient. Therefore, some heuristic indicator can make this process more efficient. For example, since $||A - \lambda_{\min}(A)I|| \leq ||A|| + \lambda_{\min}(A)$, we know that $x(\lambda^*) \geq \frac{||a||}{||A||+\lambda_{\min}(A)}$. Consequently, if we check that $\frac{||a||}{||A||+\lambda_{\min}(A)} \geq s$, we know that it is likely to be the hard case (case 1). Notice that this is just a heuristic estimate because the deflation also has effect on A and $x(\lambda^*)$, but it is not considered in the indicator $\frac{||a||}{||A||+\lambda_{\min}(A)}$.*

3. *α_i is the scalar used for the deflation. In fact, as long as α_i is significantly greater than 0, the eigenvector v_i (corresponding to $\lambda_{\min}(A)$) is fully deflated, and α_i becomes a new eigenvalue.*

6 Other Improvements and Analysis of the Algorithm

6.1 Rotation Modification

The RW algorithm is aiming at large-scale sparse trust region subproblems, which involves complicated setups and sophisticated techniques in order to fully exploit the structure and sparsity of the problem. However, if the trust region subproblem is dense and small, all these techniques may be excessive and therefore resulting in inefficiency. A more straight forward method can be used to solve such small and dense problems.

Ben-Tal and Teboulle (see [?]) have shown that all trust region subproblems can be reformulated as a TRS with diagonal matrix by double duality. However, a much simpler approach can also lead to the same result. Rotation (Diagonalization) has been widely used in the TRS to prove theoretical results (see [?, ?]). However, because a complete spectral decomposition is usually expensive, it is still not used in any algorithm. But when the problem is dense and small, we can still take advantage of this technique.

Let $A = Q\Lambda Q^T$ be the spectral decomposition of A. So we know $Q^TQ = QQ^T = I$, $\Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots \lambda_n)$ such that $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of A. Since Q is an orthogonal matrix, $\|Q^T x\| = \|x\|$. Therefore, our problem becomes:

$$\begin{array}{rll} q^* = & \min & q(x) = x^T Q\Lambda Q^T x - 2a^T x \\ & \text{s.t.} & \|x\| \leq s. \end{array} \tag{21}$$

If we let $y = Q^T x$, then $x = (Q^T)^{-1} y = Qy$, and our problem becomes:

$$\begin{array}{rll} q^* = & \min & \tilde{q}(y) = y^T \Lambda y - 2(Q^T a)^T y \\ & \text{s.t.} & \|y\| \leq s. \end{array} \tag{22}$$

which is equivalent as:

$$(TRS_{diag}) \qquad \begin{array}{rll} q^* = & \min & \tilde{q}(y) = \sum_{i=1}^n (\lambda_i y_i^2 - 2\gamma_i y_i) \\ & \text{s.t.} & \sum_{i=1}^n y_i^2 \leq s^2. \end{array} \tag{23}$$

where $y = (y_1, y_2, \ldots, y_n)^T$, $Q^T a = (\gamma_1, \gamma_2, \ldots, \gamma_n)^T$. Accordingly, the optimality conditions of (TRS_{diag}) are

$$\begin{array}{c} (\lambda_i - \lambda^*) y_i^* = \gamma_i, \quad i = 1, \ldots, n \\ \lambda^* \leq \lambda_i, \qquad\quad i = 1, \ldots, n \\ \lambda^* \leq 0 \\ \sum_{i=1}^n y_i^{*2} \leq s^2 \\ \lambda^* (s^2 - (\sum_{i=1}^n y_i^{*2})) = 0. \end{array} \tag{24}$$

After this decomposition, we can see that it can be solved easily by Newton's method or other algorithms (see Algorithm **??**).

Algorithm 6.1. ***Rotation Algorithm***

1. ***Initialization***
   Spectral decomposition of A: $A = Q\Lambda Q^T$.
   (a) *If $\lambda_{\min}(A) > 0$, A is positive definite, check if the global minimizer is in the trust region. If so, problem is solved, return the value $x^* = A^{-1}a$.*
   (b) *If $\lambda_{\min}(A) \leq 0$ and $\sum_{k \in \{j | \lambda_j = \lambda_{\min}(A)\}} \gamma_k^2 = 0$, the hard case holds.*
      i. *calculate $x(\lambda_{\min}(A))$, $\|x(\lambda_{\min}(A))\|$,*
      ii. *if $\|x(\lambda_{\min}(A))\| \leq s$, the hard case (case 2) holds,*
      iii. *return the value of $x^* = x(\lambda_{\min}(A)) + z$, where $z \in \mathcal{N}(A - \lambda_{\min}(A)I)$.*

2. ***Get the Starting Value of λ***
   Let $\epsilon > 0$ to be a small number, $\lambda < \lambda_{\min}(A)$.
   While $\|x(\lambda)\| < s$,

   (a) $\epsilon = \epsilon^2$,

   (b) $\lambda = \lambda_{\min}(A) - \epsilon$,

   (c) *calculate* $\|x(\lambda)\|$.

3. ***The Main Loop of the Newton's Method***
   While $\|x(\lambda)\| - s > tol$,
   $\lambda = \lambda - \frac{\|x(\lambda)\|^2 - s^2}{(\|x(\lambda)\|^2)'}$ *(i.e., a Newton step to solve $\|x(\lambda)\|^2 - s^2 = 0$).*

Still, there are a few shortcomings with this approach:

1. The orthogonal matrix Q, which is indeed comprised of the eigenspace of A, is generally a dense matrix. Therefore, the cost of storage is expensive if n is large.

2. This method does not exploit the sparsity of A, since the structure of Q does not depend on the sparsity of A.

3. Note that the diagonalization can be done by the Householder transformation. The Householder transformation is widely used for tri-diagonalization of symmetric matrices (see [**?**]). Then, characteristic polynomial can be evaluated efficiently for tri-diagonal matrices, and so eigenvalues can also be easily found.

6.2 Inverse Polynomial Interpolation

The different techniques used in the original RW algorithm to maximize $k(t)$ can guarantee the convergence, but we may further accelerate the algorithm by applying some other techniques. If we carefully examine all the methods, it is noticeable that all of them only use the information of only one or two points from the previous iterations. Therefore, it is natural to consider whether we can make use of more information from the previous iterations. This is indeed feasible by inverse polynomial interpolation (see Figure **??**).

To illustrate this method, we first recall that t^* is optimal if and only if

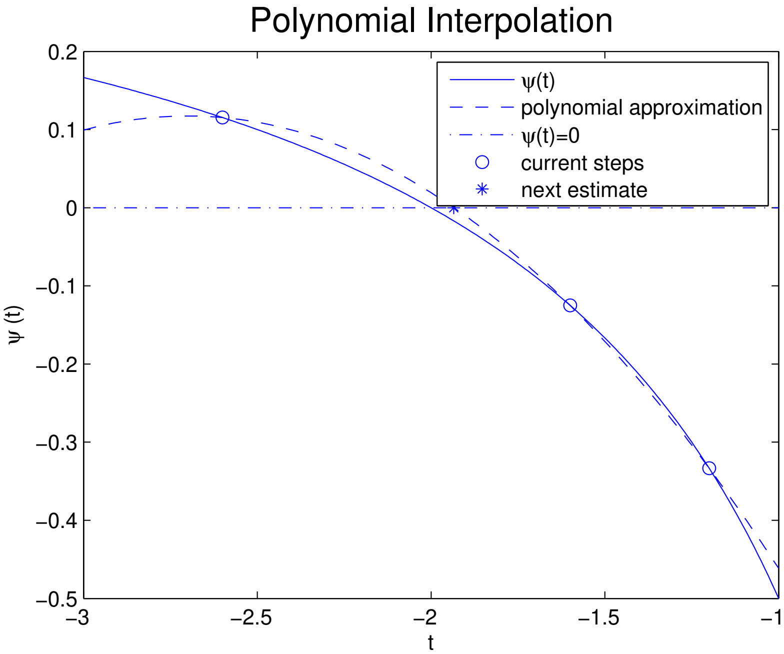
$$\phi(t^*) = \sqrt{s^2+1} - \frac{1}{y_0(t^*)} = 0, \tag{25}$$

where $y_0(t^*)$ is the first component of the eigenvector corresponding to $\lambda_{\min}(D(t))$. Although we do not have an explicit expression of t in terms of ϕ, we can estimate its value by an polynomial approximation. Precisely, assume

$$t \approx P(\phi) = c_0 + c_1\phi + c_2\phi^2 + \ldots + c_{k-1}\phi^{k-1}.$$

Therefore, if we already have k iterations, in which we obtained $\{t_j\}_{j=1,\ldots,k}$ and $\{\phi_j\}_{j=1,\ldots,k}$, where $\phi_j = \sqrt{s^2+1} - \frac{1}{y_0(t_j)}$. Let

Figure 15: Inverse Polynomial Interpolation



$$M(\phi) = \begin{pmatrix} 1 & \phi_k & \dots & \phi_k^{k-1} \\ 1 & \phi_{k-1} & \dots & \phi_{k-1}^{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1 & \dots & \phi_1^{k-1} \end{pmatrix}$$

$$\vec{c} = (c_0, c_1, \dots, c_{k-1})^T,$$

$$\vec{t} = (t_1, t_2, \dots, t_k)^T.$$

Then we can solve the linear system

$$M(\phi)\vec{c} = \vec{t}. \tag{26}$$

Then the estimated value of t such that $\phi(t) = 0$ is given by

$$t = P(0) = c_0.$$

Remark 6.1. *1. In order to guarantee that the estimated solution is stable, i.e., to make sure that it is actually an interpolation rather than an extrapolation, we need points on both the good side and the bad side of our objective. Precisely, we need to check both* $\min\{\phi_j\}_{j=1,\dots,k} < 0$ *and* $\max\{\phi_j\}_{j=1,\dots,k} > 0$ *hold, and so the result is reliable. Otherwise, it may lead to an extremely inaccurate estimated point.*

2. *Since $\phi(t)$ is a strictly decreasing function, as long as there is no same t in any two iterations, the values of $\{\phi_j\}_{j=1,\dots,k}$ are all different, guaranteeing that the matrix $M(\phi)$ is invertible, and so the linear system (**??**) has a unique solution.*

3. *It is unlikely that the initial value of t is a good (or even approximate) solution to the equation (**??**). Therefore, the absolute value of $\phi_1 = \phi(t_1)$ is generally large. Consequently, as k increases, the value of ϕ_1^{k-1} may become extremely large, rendering the matrix $M(\phi)$ ill-conditioned. Hence in each iteration, it is necessary to check the condition number of $M(\phi)$. If it is larger than a threshold, then we get rid of the information from the earliest iteration. Precisely, we delete the last row and last column of $M(\phi)$, as well as the last component of $\vec{c}$ and $\vec{t}$ respectively.*

4. *Another question is, since there are so many different functions we are interested in, (e.g., $k(t)$, $k'(t)$, $\phi(t)$ etc.) why is $\phi(t)$ the one selected to perform the inverse polynomial interpolation? Based on simple observation, we can find that $\phi(t)$ is not only strictly decreasing but also a concave function for all $t \in \mathbb{R}$. Moreover, we need to select a function whose value is predetermined for the optimal value, and we know that $\phi(t^*) = 0$ always holds. However, if we choose k(t) to perform the inverse polynomial interpolation, even if we know that $t = P(k)$, it is still difficult to find the value of t^*, because we do not know the value of $k^* = k(t^*)$, until we obtain the optimal solution.*

6.3 Approximate Eigenvalue Calculation

Although the TRS is reformulated as a parametric eigenvalue problem, we do not need the exact eigenpair in each iteration. In order to make the algorithm faster and more efficient, we may use the approximate eigenvalue estimation instead of the exact solution in each iteration.

However, when our interval is getting smaller, (i.e., when the upper bound and the lower bound of t is getting close) we need more accurate eigenvalue and eigenvector calculations to make progress with the algorithm. If the eigenvalue is inaccurate while $(t_{up} - t_{low})$ is small, the next estimation of t is probably not in (t_{low}, t_{up}), which means no progress can be made. Moreover, if the interval of the t value is small, it means that in each iteration, the value t does not change a lot, which also leads to a comparatively small change of the eigenvalue and the eigenvector. Therefore, since we are using the eigenvector obtained from the last iteration to start the Lanczos method for solving the eigenpair, the smaller the change of the eigenvector is, the faster the Lanczos method is. Therefore, it is reasonable to set the tolerance of the Lanczos method proportional to the interval of t. That is,

$$\text{tol}_{\,eigenvalue} = \tau(t_{up} - t_{low}).$$

When we use the approximate eigenvalue calculation, since the eigenvalue and the estimate for t is less accurate in each iteration, this may result in more iterations to obtain the optimal solution. However, compared to the original RW algorithm, the cost in each iteration is significantly reduced. In fact, the numerical results show that the total number of matrix-vector multiplications is also significantly reduced.

6.4 Sensitivity Analysis

The sensitivity analysis on the TRS in the easy case has been done by Flippo and Jansen in [**?**]. We can extend the results to the hard case (case 1) using Theorem **??**, below. We first introduce a lemma which gives an easier way to calculate the optimal objective value of the TRS.

Lemma 6.1. *If (x^*, λ^*) is an optimal primal-dual solution pair to the TRS, then*

$$q^* = \lambda^* s^2 - a^T x^*.$$

Proof: If (x^*, λ^*) is an optimal primal-dual solution pair to the TRS, then the optimality conditions imply that:

$$(A - \lambda^* I)x^* = a,$$

$$\lambda^*(x^{*T} x^* - s^2) = 0.$$

Therefore,

$$\begin{aligned} q^* &= x^{*T} A x^* - 2a^T x^* \\ &= x^{*T}(\lambda^* x^* + a) - 2a^T x^* \\ &= \lambda^* s^2 - a^T x^*. \end{aligned}$$

■

We now obtain sensitivity analysis on the easy case and the hard case (case 1).

Theorem 6.1. *Consider the perturbed TRS:*

$$(TRS_u) \qquad \begin{array}{rll} q^*(u) = & \min & q(u,x) := x^T A(u) x - 2a(u)^T x \\ & s.t. & \|x\| \le s(u), \end{array} \tag{27}$$

where $u \in \mathbb{R}$ is the perturbation parameter, and $A(u)$, $a(u)$ and $s(u)$ are differentiable functions of u. If the solution to this problem is $(x^*(\bar{u}), \lambda^*(\bar{u}))$ when $u = \bar{u}$, and we know that the easy case or the hard case (case 1) holds for this problem, then

$$\frac{dq^*(\bar{u})}{du} = -2x^*(\bar{u})^T \frac{da(\bar{u})}{du} + x^*(\bar{u})^T \frac{dA(\bar{u})}{du} x^*(\bar{u}) + 2\lambda^*(\bar{u}) s(\bar{u}) \frac{ds(\bar{u})}{du}.$$

Proof: First of all, since the easy case or the hard case (case 1) holds, we know that $\lambda^*(\bar{u}) < \lambda_{\min}(A(\bar{u}))$ and $x^*(\bar{u})$ is unique. According to the optimality conditions, we have

$$(A(\bar{u}) - \lambda^*(\bar{u}) I) x^*(\bar{u}) = a(\bar{u}), \tag{28}$$

$$\lambda^*(\bar{u})(x^*(\bar{u})^T x^*(\bar{u}) - s(\bar{u})^2) = 0. \tag{29}$$

From Lemma **??**, we have

$$q^*(\bar{u}) \quad = \lambda^*(\bar{u}) s(\bar{u})^2 - a(\bar{u})^T x^*(\bar{u}) \tag{30}$$

Differentiating (**??**) with respect to u, we can obtain

$$\frac{dq^*(\bar{u})}{du} = \frac{d\lambda^*(\bar{u})}{du} s(\bar{u})^2 + 2s(\bar{u})\lambda^*(\bar{u}) \frac{ds(\bar{u})}{du} - \frac{da(\bar{u})}{du}^T x^*(\bar{u}) - \frac{dx^*(\bar{u})}{du}^T a(\bar{u}). \tag{31}$$

Also, differentiating (**??**), we have

$$(\frac{dA(\bar{u})}{du} - \frac{d\lambda^*(\bar{u})}{du} I) x^*(\bar{u}) + (A(\bar{u}) - \lambda^*(\bar{u}) I) \frac{dx^*(\bar{u})}{du} = \frac{da(\bar{u})}{du}. \tag{32}$$

Left multiplying by $x^*(\bar{u})^T$ on both sides of (**??**), we can obtain

$$x^*(\bar{u})^T(\frac{dA(\bar{u})}{du} - \frac{d\lambda^*(\bar{u})}{du}I)x^*(\bar{u}) + x^*(\bar{u})^T(A(\bar{u}) - \lambda^*(\bar{u})I)\frac{dx^*(\bar{u})}{du} = x^*(\bar{u})^T\frac{da(\bar{u})}{du}. \tag{33}$$

Combine (**??**) and (**??**), and the result follows. ■

In the hard case (case 2), in order to apply the sensitivity analysis results in convex programming to the TRS, we need to reformulate the TRS to an equivalent convex problem. This can be done by the following theorem.

Theorem 6.2 ([**?**]). *If (x^*, λ^*) is the optimal solution to the TRS, and the hard case (case 2) holds, then the TRS is equivalent to the following convex programming problem:*

$$(TRS_c) \qquad \begin{array}{rll} q_c^* = & \min & q_c(x) := x^T(A - \lambda_{\min}(A)I)x - 2a^Tx + s^2\lambda_{\min}(A) \\ & s.t. & \|x\| \leq s, \end{array} \tag{34}$$

in the sense that (x^*, λ^*) solves the TRS if and only if $(x^*, 0)$ solves the TRS_c, and $q^* = q_c^*$.

Proof: Since the hard case (case 2) holds for the TRS, we know that $\lambda^* = \lambda_{\min}(A)$, $a \in \mathcal{R}(A - \lambda^* I)$, and $\|x(\lambda^*)\| \leq s$.
If $\lambda^* = \lambda_{\min}(A) = 0$, then the TRS is exactly the same as the TRS_c, and the conclusion is obviously true.
If $\lambda^* = \lambda_{\min}(A) < 0$, then we know $x^* = x(\lambda^*) + z$ with $z \in \mathcal{N}(A - \lambda^* I)$ and $\|x^*\| = s$ solves the TRS. From Lemma **??**, we have

$$\begin{aligned} q^* &= \lambda^* s^2 - a^T x^* \\ &= \lambda^* s^2 - a^T(x(\lambda^*) + z) \\ &= \lambda^* s^2 - a^T x(\lambda^*) + a^T z \\ &= \lambda^* s^2 - a^T x(\lambda^*). \end{aligned}$$

The last step is because we know that the hard case (case 2) holds, and so $a \in \mathcal{R}(A - \lambda^* I)$, and we also know that $z \in \mathcal{N}(A - \lambda^* I)$. Therefore, $a \perp z$ and $a^T z = 0$. Also, according to Lemma **??** Item 3, $(x(\lambda^*), 0)$ solves the TRS_c, and according to Lemma **??**, we have

$$q_c^* = 0s^2 - a^T x(\lambda^*) + s^2\lambda^* = \lambda^* s^2 - a^T x^*.$$

Therefore, $q^* = q_c^*$. Also, $\|x^*\| \leq s$, and so x^* is a feasible solution to the TRS_c. Since

$$\begin{array}{rcl} q_c(x^*) & = & x^{*T}(A - \lambda_{\min}(A))x^* - 2a^T x^* + s^2\lambda_{\min}(A) \\ & = & x(\lambda^*)^T(A - \lambda_{\min}(A)I)x(\lambda^*) + 2x(\lambda^*)^T(A - \lambda_{\min}(A)I)z \\ & & +z^T(A - \lambda_{\min}(A)I)z - 2a^T x(\lambda^*) - 2s^T z + s^2\lambda_{\min}(A) \\ & = & x(\lambda^*)^T(A - \lambda_{\min}(A)I)x(\lambda^*) - 2a^T x(\lambda^*) + s^2\lambda_{\min}(A) \\ & = & q_c^*, \end{array}$$

where the last step is because $z \in \mathcal{N}(A - \lambda_{\min}(A)I)$ and $a^T z = 0$. Therefore, $(x^*, 0)$ solves the TRS_c. ■

Since the TRS in the hard case (case 2) can be reformulated as a convex problem, we have the following result on the sensitivity analysis with respect to q^*.

Corollary 6.1. *If the hard case (case 2) holds for the TRS, and $s(u) = s + u$ is perturbed. Let $q^*(u)$ be the optimal objective value, then*

$$\frac{dq^*(0)}{du} = -\lambda^*$$
$$q^*(u) \geq q^*(0) - \lambda^* u.$$

Proof: The result follows from Theorem **??** . ∎

6.5 Error Analysis on t^*

We have shown in Section **??** (also see [**?**]) that all trust region subproblems can be reformulated to a diagonal problem (i.e., the matrix A is diagonal). In this section, we assume that A is diagonal, with distinct diagonal entries (eigenvalues) $\lambda_1 < \lambda_2 < ... < \lambda_n$. Moreover, we require the hard case does not hold so that $k(t)$ and $\lambda_{\min}(D(t))$ are both twice differentiable.

Since the RW algorithm is mainly finding the optimal value of t, we analyze how the error on t affect the results. Assume $t^* = \arg\min k(t)$, then $q^* = k(t^*)$. Let $t = t^* + \Delta t$ be an inaccurate solution with a very small error Δt.

6.5.1 How Δt Affects λ^*

Since the easy case holds, recall from the proof of Theorem **??** that there is a unique λ^* in $(-\infty, \lambda_{\min}(A))$ such that $d(\lambda^*) = t^*$, where

$$d(\lambda) = \lambda + \sum_{k \in \{j | \lambda_j > \lambda\}} \frac{a_k^2}{\lambda_k(A) - \lambda}.$$

Moreover,

$$d'(\lambda) = 1 + \sum_{k \in \{j | \lambda_j > \lambda\}} \frac{a_k^2}{(\lambda_k(A) - \lambda)^2} > 0,$$

and so $d(\lambda)$ is a monotonically increasing function. If $t = t^* + \Delta t$, then $d(\lambda^* + \Delta\lambda) = t$. Since Δt is very small, we may assume that

$$d'(\lambda)\Delta\lambda = \Delta t + o(\Delta t),$$

so

$$\begin{aligned} |\Delta\lambda| &= \left|\frac{\Delta t + o(\Delta t)}{d'(\lambda^*)}\right| \\ &= \left|\frac{\Delta t + o(\Delta t)}{1 + \sum_{k \in \{j|\lambda_j > \lambda\}} \frac{a_k^2}{(\lambda_k - \lambda^*)^2}}\right| \\ &\leq \left|\frac{\Delta t + o(\Delta t)}{1}\right| \\ &= |\Delta t + o(\Delta t)|. \end{aligned}$$

Another approach will also leads us to the same result. Since $\lambda^* = \lambda_{\min}(D(t^*))$,

$$\begin{aligned}
|\Delta\lambda| &= |\lambda - \lambda^*| \\
&= |\lambda_{\min}(D(t)) - \lambda_{\min}(D(t^*))| \\
&= |\lambda'_{\min}(D(t^*))(t - t^*)| \\
&= |(y_0^*)^2 \Delta t + o(\Delta t)| \\
&\leq |(y_0^*)^2||\Delta t + o(\Delta t)| \\
&\leq |\Delta t + o(\Delta t)|.
\end{aligned}$$

Therefore, the value of λ is stable for t.

6.5.2 How Δt Affects k^*

Also, we want to see how does this affect the value of $k(t)$. When $\Delta t > 0$, $t > t^*$ and $-1 < k'(t) < 0$, then $\Delta k < 0$ and

$$\begin{aligned}
\Delta k &= k(t) - k(t^*) \\
&> -1 \times \Delta t \\
&= -\Delta t.
\end{aligned}$$

So $0 > \Delta k > -\Delta t$ when $\Delta t > 0$.

If $\Delta t < 0$, then $t < t^*$ and $0 < k'(t) < s^2$, so we have $\Delta k > 0$ and

$$\begin{aligned}
\Delta k &= k(t) - k(t^*) \\
&< -s^2 \times \Delta t.
\end{aligned}$$

So $-s^2\Delta t > \Delta k > 0$ when $\Delta t < 0$.

Therefore, We can conclude that $|\Delta k| < \max\{|\Delta t|, s^2|\Delta t|\} < (s^2+1)|\Delta t|$. Since $q^* = k(t^*)$, the value of k is also stable for t, as long as the value of s is not large. In other words, the trust region being too large may result in an unstable value of the quadratic objective function.

6.5.3 How Δt Affects $k'(t^*)$

Also, if $y^i(t) = \begin{pmatrix} y_0^i(t) \\ w^i(t) \end{pmatrix}$ is the corresponding eigenvector of $\lambda_i(D(t))$. In particular, $y(t) = \begin{pmatrix} y_0(t) \\ w(t) \end{pmatrix}$ is the corresponding eigenvector of $\lambda_{\min}(D(t))$. Then from [?], the second derivative of $k(t)$ can be calculated by:

$$k''(t) = 2(s^2+1)(\sum_i \frac{(y_0 y_0^i)^2}{\lambda_{\min}(D(t)) - \lambda_i(D(t))}).$$

Therefore,

$$\begin{aligned}
|\Delta k'(t)| &= |k''(t^*)\Delta t + o(\Delta t)| \\
&= |2(s^2+1)(\textstyle\sum_i \frac{(y_0 y_0^i)^2}{\lambda_{\min}(D(t))-\lambda_i(D(t))})\Delta t + o(\Delta t)| \\
&\leq |2(s^2+1)(\textstyle\sum_i \frac{1}{\lambda_{\min}(D(t))-\lambda_i(D(t))})||\Delta t| + |o(\Delta t)|.
\end{aligned}$$

As a result, we can see that $k'(t)$ is also stable for t given the condition that the second smallest eigenvalue of $D(t^*)$ is well separated from $\lambda_{\min}(D(t^*))$. On the other hand, if $\lambda_{\min}(D(t^*)) \approx \lambda_2(D(t^*))$, $\Delta|k'(t)|$ may become very large even if t is only a little bit perturbed from t^*.

Recall that $\lambda_{\min}(D(t^*)) \leq \lambda_{\min}(A) \leq \lambda_2(D(t^*))$. Since $d(\lambda_{\min}(D)) = t^*$ where

$$d(\lambda) = \lambda + \sum_{k\in\{j|\lambda_j>\lambda\}} \frac{(a_k)^2}{\lambda_k(A)-\lambda},$$

$$d'(\lambda) = 1 + \sum_{k\in\{j|\lambda_j>\lambda\}} \frac{(a_k)^2}{(\lambda_k(A)-\lambda)^2} > 0,$$

$$d''(\lambda) = \sum_{k\in\{j|\lambda_j>\lambda\}} \frac{2(a_k)^2}{(\lambda_k(A)-\lambda)^3}.$$

So when $\lambda \in (-\infty, \lambda_1)$, $d(\lambda)$ is convex and strictly increasing and its range is $(-\infty, \infty)$. Also, we know that $d(\lambda^*) = t^*$. Therefore, in the easy case, as long as t^* is not large, $d(\lambda^*)$ is also not large, and so $\lambda_{\min}(D)$ is well separated from $\lambda_{\min}(A)$, and therefore also well separated from $\lambda_2(D)$. As a result $k'(t)$ is also stable for t.

6.5.4 How Δt Affects q^*

Since the value of x is obtained by

$$x(t) = \frac{w(t)}{y_0(t)}$$

and

$$q(x) = x(t)^T A x(t) - 2a^T x(t).$$

Let

$$x'(t) = \begin{pmatrix} x_1'(t) \\ x_2'(t) \\ \vdots \\ x_n'(t) \end{pmatrix} \quad w'(t) = \begin{pmatrix} w_1'(t) \\ w_2'(t) \\ \vdots \\ w_n'(t) \end{pmatrix}.$$

Then

$$\begin{aligned}
\Delta q &= q(t) - q(t^*) \\
&= q'(t^*)\Delta t + o(\Delta t) \\
&= 2(Ax(t^*) - a)^T x'(t^*)\Delta t + o(\Delta t) \\
&= 2(Ax(t^*) - a)^T (\tfrac{w'(t^*)}{y_0(t^*)} - \tfrac{w(t^*)y_0'(t^*)}{(y_0(t^*))^2})\Delta t + o(\Delta t). \\
&= 2(\lambda^* x^*)^T (\tfrac{w'(t^*)}{y_0(t^*)} - \tfrac{w(t^*)y_0'(t^*)}{(y_0(t^*))^2})\Delta t + o(\Delta t).
\end{aligned}$$

Therefore, we know that when $\|x^*\|$ or λ^* is close to 0, $q(x)$ is still stable to t. However, if $y_0(t^*)$ is small, the value of $q(x)$ is sensitive to t.

7 Application to Unconstrained Optimization

The trust region method is a widely used optimization method for unconstrained optimization problems, e.g., [?]. It is an iterative method which calculates a step to improve the value of the objective function in each iteration, subject to the condition that the step is within a proper trust region. On the other hand, the trust region is also adjusted adaptively according to the performance at the current iteration. Generally, we require the objective function to be twice differentiable so that in each iteration we can obtain the gradient and the Hessian matrix of the objective function, which are used in the trust region subproblem.

Like Newton's method, it is also a second order method, while methods like steepest descent only require the information of the first derivative. However, it is different from Newton's method in the following two aspects. (i) One of the biggest advantage of Newton's method is that it is scale-free, while the trust region method is not; (ii) sparsity can be easily exploited in Newton's method while most classic approaches for the TRS cannot take advantage of the sparsity or the structure of the Hessian matrix.

We first present the standard trust region method. In each iteration, if the estimate from the last iteration is x_k, then the trust region subproblem for the current iteration is defined as:

$$q^* = \begin{array}{ll} \min & q(v) := v^T H v + 2g^T v \\ \text{s.t.} & \|v\| \le s, \end{array} \tag{35}$$

where

$$H = \nabla^2 f(x_k), \quad g = \nabla f(x_k).$$

The motivation behind this is the Taylor expansion for $f(x_k + v)$:

$$f(x_k + v) \approx f(x_k) + \nabla f(x_k)^T v + \frac{1}{2} v^T \nabla^2 f(x_k) v.$$

Since this is true only if v is relatively small, the radius of the trust region s determines how much we "trust" the model. It is adjusted by the performance of the model, which is measured by:

$$\rho_k = \frac{f(x_k + v) - f(x_k)}{q(v)}.$$

If ρ_k is close to 1 (or greater than 1), then we know that the model represents the function quite well, and so we should take the step and the trust region can be expanded in the next iteration. However, if ρ_k is small or even negative, the trust region is too large and so it has to shrink in the next iteration.

Since we have the information of the Hessian and the gradient in each iteration, they can also be used as stopping criteria. Precisely, the algorithm terminates when the Hessian is positive semi-definite and the gradient is sufficiently small, which means the minimum is obtained. Notice that the trust region method does not guarantee convergence to the global minimum.

The algorithm can be summarized as Algorithm **??**: (see [**?**])

Algorithm 7.1. ***Trust Region Method***
Set x_0, s_0, $0 < \alpha_1 \leq \alpha_2 < 1$, $0 < \beta_1 \leq \beta_2 < 1$.
While H_k *is not positive semi-definite or* $\|g_k\| \geq \mathrm{tol}_g$,

1. *solve the trust region subproblem:*

$$(TRS_k) \qquad \begin{array}{rll} q^* = & \min & q(v_k) := v_k^T H_k v_k + 2 g_k^T v_k \\ & s.t. & \|v_k\| \leq s_k. \end{array} \tag{36}$$

2. *Measure the performance of* v_k*:*

$$\rho_k = \frac{f(x_k + v_k) - f(x_k)}{q(v_k)}.$$

3. *Update:*
   (a) *If* $\rho_k \geq \alpha_2$, $s_{k+1} \in (s_k, +\infty)$. *(trust region expands)*
   (b) *If* $\alpha_2 > \rho_k \geq \alpha_1$, $s_{k+1} \in (\beta_2 s_k, s_k)$.
   (c) *If* $\rho_k < \alpha_1$, $s_{k+1} \in (\beta_1 s_k, \beta_2 s_k)$. *(trust region shrinks)*
   (d) *If* $\rho_k > \alpha_1$, $x_{k+1} = x_k + v_k$. *(take the step)*
   (e) $H_{k+1} = \nabla^2 f(x_{k+1})$, $g_{k+1} = \nabla f(x_{k+1})$.

7.1 Modification for the Hard Case

Recall that when the hard case (case 2) occurs in the trust region subproblem, the optimal solution is given by $x^* = x + z$ where $x = (A - \lambda_{\min}(A)I)^\dagger a$ and $z \in \mathcal{N}(A - \lambda_{\min}(A)I)$, such that $\|z\|^2 = s^2 - \|x\|^2$. Therefore, whenever the hard case (case 2) occurs,

1. The solution has to be "pushed" to the boundary in order to satisfy the optimality conditions.

2. Optimal solution is not unique, since z is not unique.

3. Though the objective value of the trust region subproblem is the same for different optimal solutions, it's not the case of the objective function in the unconstrained optimization problem. i.e., $f(x_k + v_k)$ is different for different v_k.

Therefore, when the hard case (case 2) occurs, the performance of the model is usually not good since the trust region subproblem is indeed unstable. A common technique to prevent the hard case (case 2) and make the solution to the TRS unique is to decrease the value of s, and when $s < \|x\|$, the hard case (case 2) does not occur. From another angle, when the hard case (case 2) occurs, s is too large and we trust the model too much. Therefore, instead of the standard routine in the trust region method, a better approach is to adjust the trust region in order to make the solution to the TRS more stable. Precisely,

$$v_k = (H_k - \lambda_{\min}(H_k))^\dagger g_k,$$

$$s_{k+1} = \|(H_k - \lambda_{\min}(H_k))^\dagger g_k\|.$$

7.2 Warm Start

Notice that when ρ_k is small or negative, the step is not taken and so $x_{k+1} = x_k$, which also results in $H_{k+1} = H_k$, $g_{k+1} = g_k$. Therefore, the (TRS_k) and the (TRS_{k+1}) are the same except for the change of s, the trust region radius. It is inefficient to solve the almost same problem all over again. Therefore, we use the warm start when the step is not taken in an iteration.

We keep the smallest eigenvalue and its corresponding eigenvector of H_k, which can all be used in (TRS_{k+1}). In addition, because $D(t) = \begin{pmatrix} t & -a^T \\ -a & A \end{pmatrix}$ is independent to s, all the smallest eigenvalues and eigenvectors of $D(t)$ for different t can be saved for further use. Given the value of $\lambda_{\min}(D(t))$ and $y(t)$, we know that $k(t)$ and $k'(t)$ can also be easily obtained with little computation. Then we pick t_0, which has the greatest value of $k(t)$ (or smallest magnitude of $k'(t)$) as the starting point of the new iteration. Details are shown in Algorithm **??**:

Algorithm 7.2. ***Warm Start***
Let t_1, t_2, ... t_m are the iterative estimating points obtained in the process of solving (TRS_k). The values of $\lambda_{\min}(D(t_1))$, $\lambda_{\min}(D(t_2))$, ... ,$\lambda_{\min}(D(t_m))$ and $y_0(t_1)$, $y_0(t_2)$, ... ,$y_0(t_m)$ are saved.
If $\rho_k < \alpha_1$, then $x_{k+1} = x_k$, $H_{k+1} = H_k$, $g_{k+1} = g_k$.

1. *Compute*
$$t_0 = \arg \max_{i=1,2,\ldots m} k(t_i) = (s_{k+1}^2 + 1)\lambda_{\min}(D(t_i)) - t_i,$$
or
$$t_0 = \arg \min_{i=1,2,\ldots m} |k'(t_i)| = |(s_{k+1}^2 + 1)y_0(t_i)^2 - 1|.$$

2. *Solve (TRS_{k+1}) using t_0 as the starting point.*

8 Numerical Experiments

In this section, we test the performance of the improvements discussed above. We first compare the performance of the RW algorithm solving the TRS and the scaled TRS with the scalar $r = s$. Then, we test the results of the shift and deflation in the hard case (case 2). Then the performance of the improvements and techniques we discussed before are also tested, including the rotation modification, the inverse polynomial interpolation and the approximate eigenvalue calculation. Since the hard case (case 2) can be efficiently solved by the shift and deflation, and the hard case (case 1) actually can be treated by the same method as the easy case, we only perform the tests in the easy case for all the techniques and improvements except the shift and deflation. We then compare the revised RW algorithm with the LSTRS, a TRS solver written by Rojas, Santos, Sorenson and Voss, (see [**?**, **?**]). Finally, results on CUTEr problem test set (see [**?**] and [**?**]) are also given when the TRS algorithm is used in a trust region method to solve unconstrained optimization problems.

All the tests are done using MATLAB 2011a in the Windows 7 Professional 64-bit environment. The computer is a laptop with a dual core 2.66GHz CPU and 4GB Ram. Each point on the figures is the average of 20 test problems. Unless otherwise specified, the density of the matrix in each problem is 0.01. Also, the tolerance of the first order optimality condition is 10^{-8} for the TRS. The random TRS are generated using the following MATLAB Codes:

```
% % Generate an easy case problem
Ae=sprandsym(n,density);
ae=randn(n,1);
se=abs(randn);
% % Generate a hard case (case 2) problem
Ah=sprandsym(n,density); % generate a random symmetric Ah
(v,lambda)=eigs(Ah,1,'SA'); % the smallest eigenvalue of Ah
xtempopt=randn(n,1); % generate a random optimal solution
sh=1.1*norm(xtempopt); % trust region radius
ah=Ah*xtempopt -lambda*xtempopt; % set ah = (Ah-lambda I)xtempopt
```

8.1 Scaling

Recall that for a given scalar $r > 0$, we can scale the TRS to obtain an equivalent problem using

$$A \leftarrow r^2 A, \quad a \leftarrow ra, \quad s \leftarrow \frac{1}{r}s.$$

From Figure **??**, we can see that the scaling can considerably improve the performance of the RW algorithm. Especially when the size of the problem n is getting larger, the difference of the matrix-vector multiplications for the original TRS and the scaled TRS is also getting bigger. Therefore, setting $r = s$ certainly can accelerate the RW algorithm.

8.2 Shift and Deflation

The shift and deflation is another technique which we may use to improve the performance of the algorithm. From Figure **??**, we can see that the performance of the shift and deflation is remarkable. As we expect, if the multiplicity of the smallest eigenvalue of A is 1, then it solves the TRS by calculating 2 eigenvalues (to make sure the smallest eigenvalue is not multiple) and solving a well-conditioned linear system. This is certainly much faster than the original RW algorithm, which needs to calculate an eigenpair in each iteration. Also, notice that the shift and deflation is not used at all in the easy case, because it is only triggered when $|v^T a|$ is sufficiently small, where v is the eigenvector corresponding to the smallest eigenvalue of A. Therefore, it does not affect the speed at all in the easy case.

Figure 16: Scaling; Varying n; Density= 0.01; Easy Case

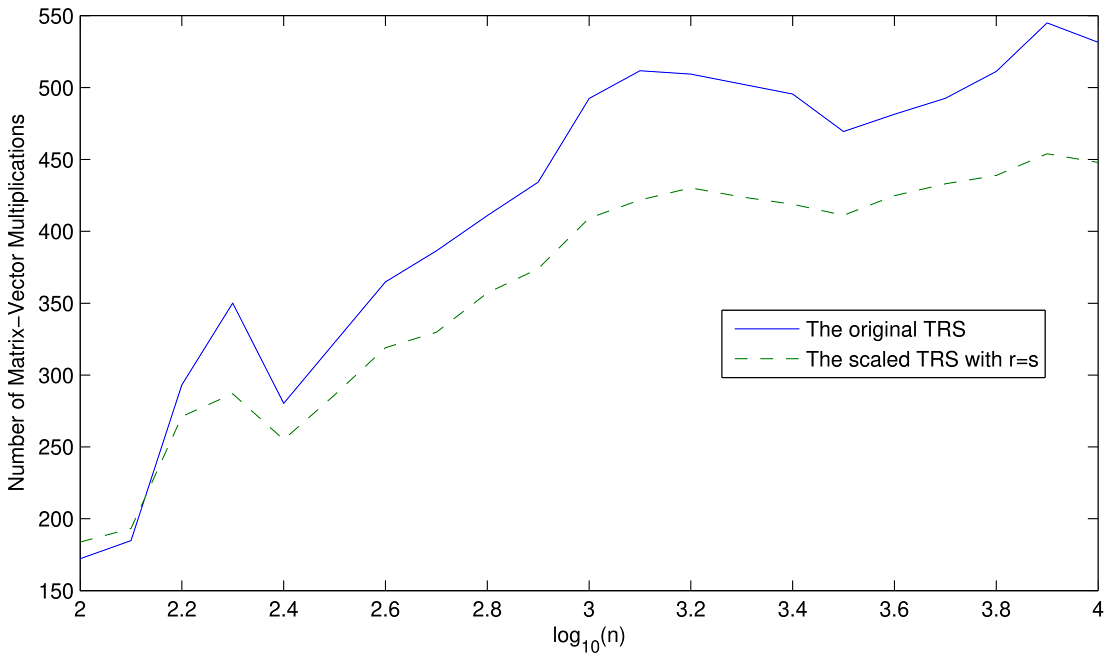
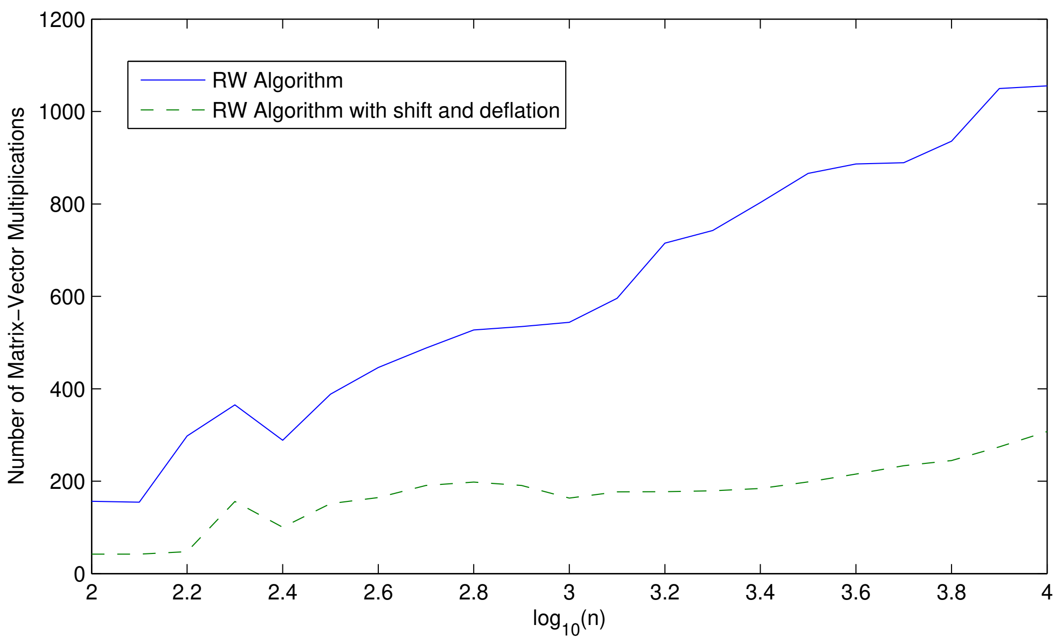


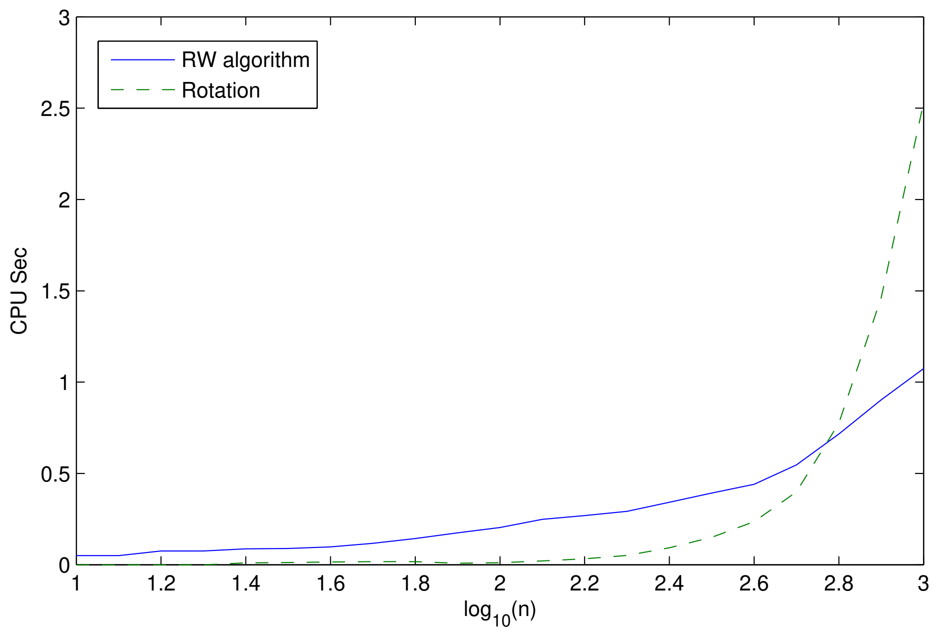
Figure 17: Shift and Deflation; Varying n; Density= 0.01; Hard Case (Case 2)



8.3 Rotation Versus the RW Algorithm

In this section, we test the rotation algorithm from Section **??**, and compare it with the RW algorithm. In the first figure (see Figure **??**), we fix the density at 0.01 and solve some randomly generated trust region subproblems with the same density but different sizes n.

Figure 18: Rotation vs RW; Varying n; Density= 0.01; Easy Case



We can see that the time of the RW algorithm is increasing slowly with respect to the problem size, while the rotation algorithm is much more sensitive to the problem size. The reason is also obvious —— the complete spectral decomposition becomes much more expensive when the matrix size is larger. However, when the size of the problem is small ($n \leq 10^{2.6} \approx 400$), in both the easy and the hard cases, the rotation algorithm costs much less than the RW algorithm, even though the problem is sparse.

We next fix the size of the problems ($n = 1000$), and compare the two algorithms by solving problems with different densities (see Figure **??**).

As we expect, when the problem is sparse, the RW algorithm performs much better. However, when the density is getting higher, since the rotation algorithm does not take advantage of the sparsity at all, we conclude that it is not affected by the higher density. Therefore, the cost of time for the rotation remains almost the same for different densities.

In sum, the rotation algorithm is more efficient when the problem is dense and small. The smaller size and the more dense it is, the better it performs comparing to the RW algorithm.

8.4 Inverse Polynomial Interpolation

Inverse polynomial interpolation uses a high order polynomial to approximate the inverse function of $\phi(t)$ in order to get a better approximation of $t(\phi)$. In the easy case (see Figure **??**),

Figure 19: Rotation vs RW; Varying Density; $n = 1000$; Easy Case

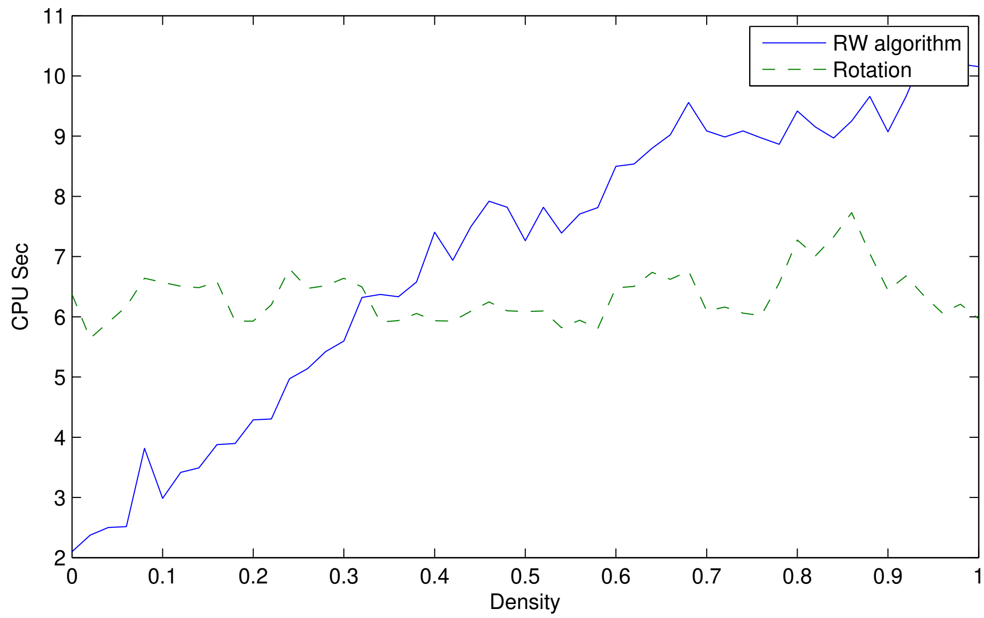
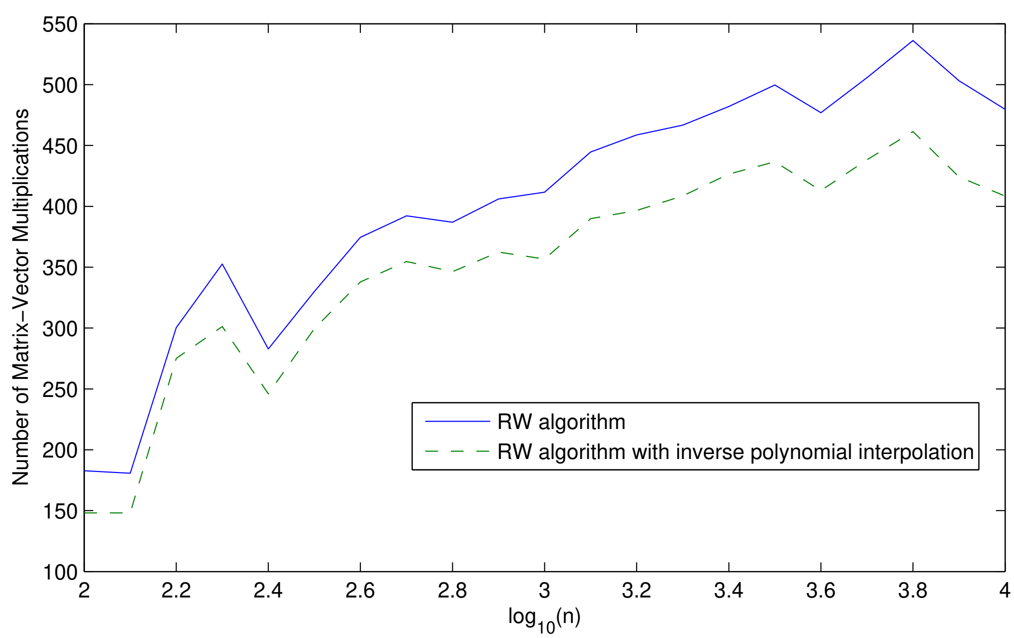


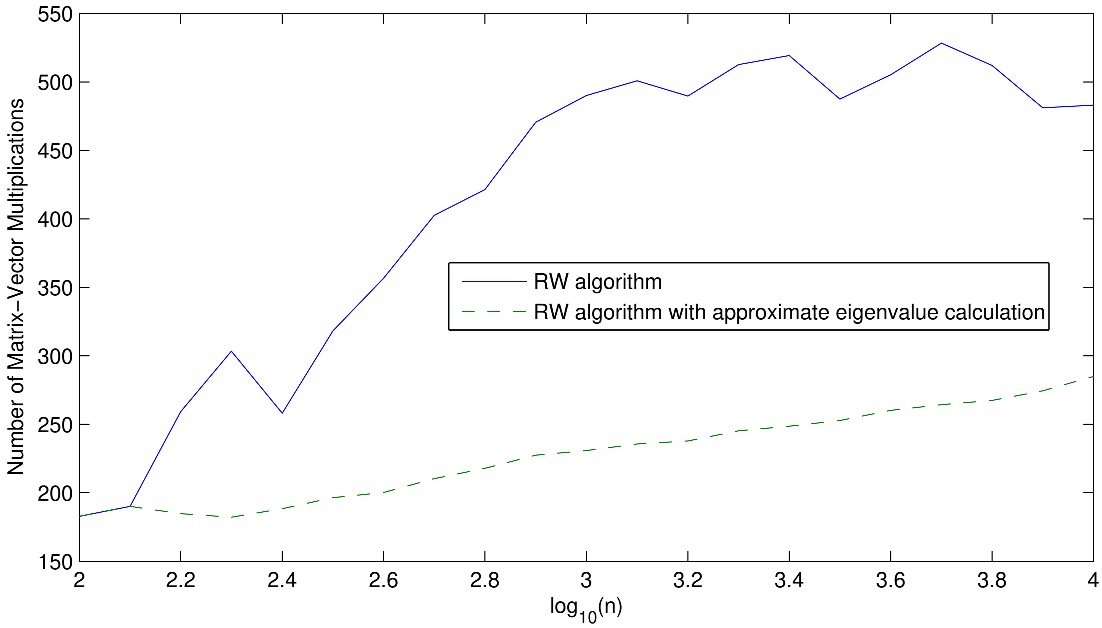
Figure 20: Inverse Polynomial Interpolation; Varying n; Density= 0.01; Easy Case



we can see that this technique does give some improvement to the algorithm. The improvement mainly comes from less iterations required to obtain t^* because the estimate is more precise in each iteration.

8.5 Approximate Eigenvalue Calculation

Figure 21: Approximate Eigenvalue Calculation; Varying n; Density= 0.01; Easy Case



Approximate eigenvalue calculation is indeed a very simple idea, in which we decrease the accuracy of eigenvalue calculation in each iteration but instead increase the number of iterations as a tradeoff. Note that the accuracy of the TRS actually remains the same, because none of the stopping criteria to terminate the algorithm are changed.

From the figure (see Figure **??**), we know that in the easy case, we have significant improvements comparing to the original RW algorithm. We also want to mention that in the tests we dynamically change the accuracy of the eigenvalue calculation in each iteration, which turns out to perform well. Precisely, as the estimates are approaching the solution, tolerance has to be set smaller and smaller. If the accuracy is fixed, it is the same as the original RW algorithm (when the accuracy is very high in each iteration) or it fails to converge (when the accuracy is too low).

8.6 Revised RW Algorithm Versus LSTRS

The LSTRS (see [**?**, **?**]) is also a TRS solver which is designed for large-scale sparse problems. It reformulates the TRS into a parametric eigenvalue problem. It is also matrix-free in the sense that the matrix A is only used as an operator to perform matrix-vector multiplications. However, comparing to the RW algorithm, it uses a different interpolation scheme to update the parameter. We now compare the revised RW algorithm with the LSTRS in solving the TRS in both the easy case and the hard case.

Figure 22: Revised RW vs LSTRS; Varying n; Density= 0.01; Easy Case

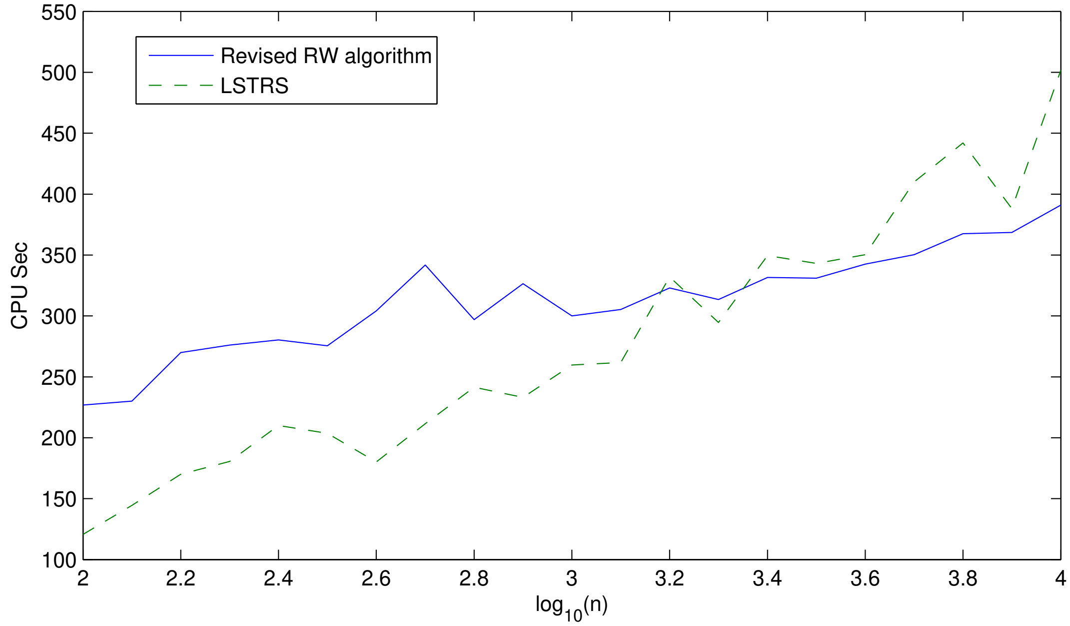
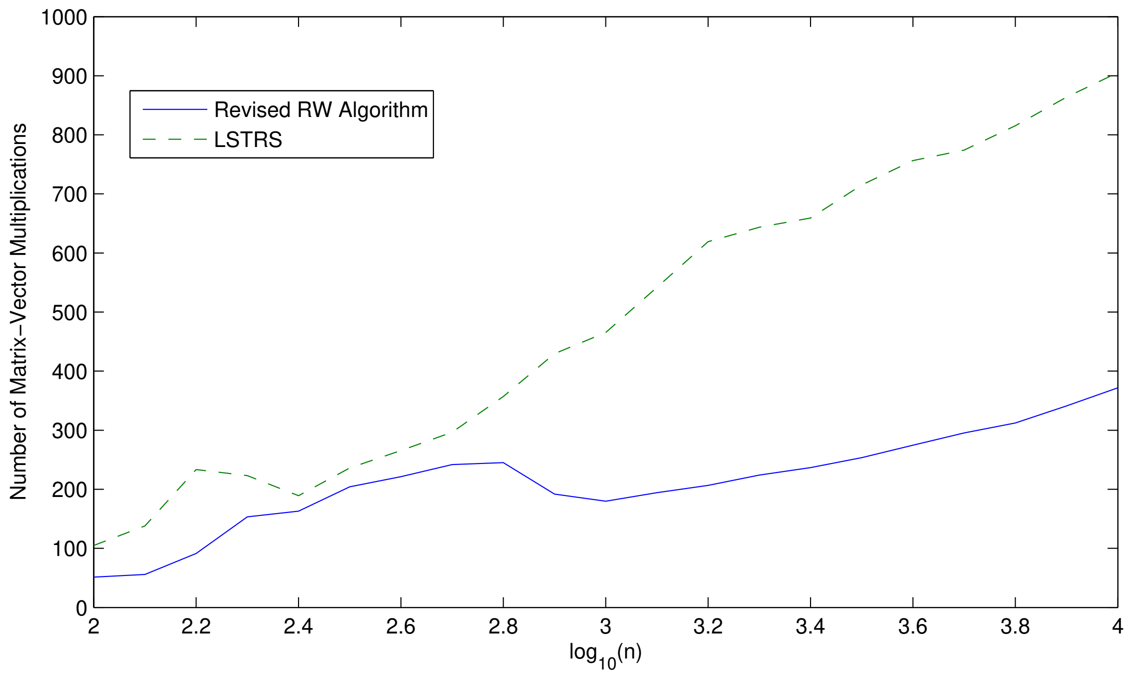


Figure 23: Revised RW vs LSTRS; Varying n; Density= 0.01; Hard Case (Case 2)



From Figure **??**, we can see that the LSTRS is more sensitive to the problem size than the revised RW algorithm in the easy case. From an average of 20 randomly generated problems in each problem size n, the LSTRS takes 124.10 matrix-vector multiplications to solve a problem when $n = 100$ and it takes 498.75 matrix-vector multiplications to solve a problem when $n = 10000$. But for the revised RW algorithm, the number of matrix-vector multiplications increases from 228.25 to 391.70 when n increases from 100 to 10000. Therefore, the LSTRS performs better when the problem size is small, while the revised RW algorithm is better when the problem size is large.

In the hard case (case 2), from Figure **??** we can see that the revised RW algorithm outperforms the LSTRS. This is mainly because of the application of the shift and deflation in the revised RW algorithm.

8.7 Hard Case (Case 2) with Multiple Smallest Eigenvalue

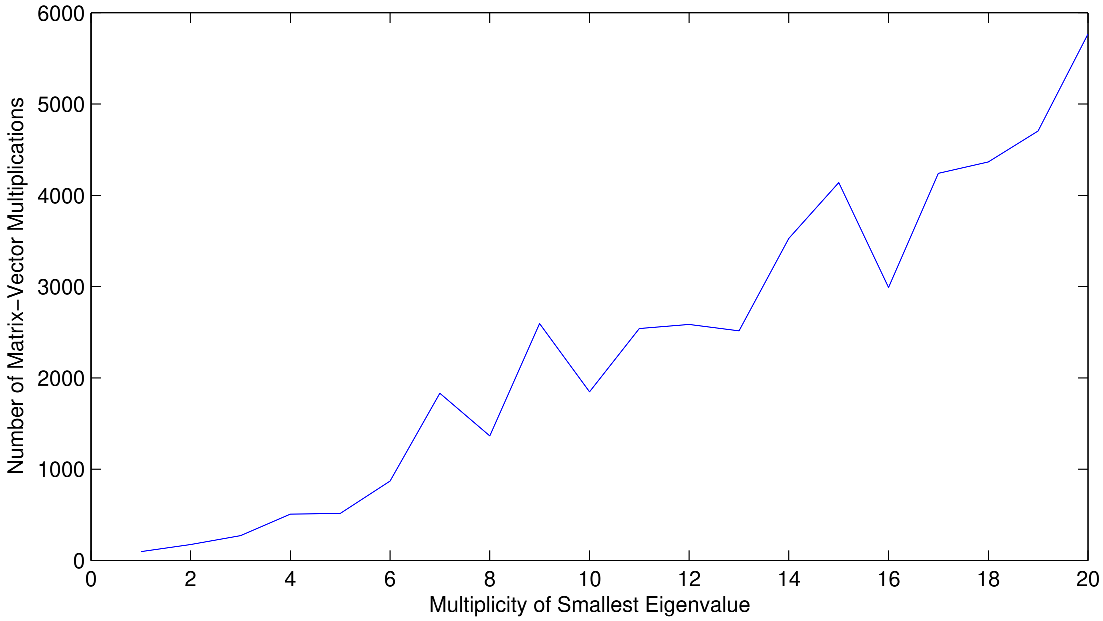
Let m be the multiplicity of the smallest eigenvalue of the matrix A. A TRS in the hard case (case 2) with $m > 1$ is generally difficult to solve. When $m > 1$, we have numerical difficulties in solving the eigenpairs using Lanczos type methods. Moreover, m eigenvectors (the basis of the eigenspace corresponding to the smallest eigenvector) have to be deflated in order to get rid of the hard case (case 2) in the shift and deflation. When m is large, the cost may become rather expensive. In this section we generate TRS in the hard case (case 2) randomly with different multiplicities of the smallest eigenvalue and see how the revised RW algorithm performs.

Firstly we show that how a TRS in the hard case (case 2) with multiple smallest eigenvalue is generated. Let n be the problem size, $m < n$ be a positive integer (the multiplicity of the smallest eigenvalue). Let $A_0 \in \mathbb{R}^{(n-m)\times(n-m)}$ be a randomly generated symmetric sparse matrix, $a_0 \in \mathbb{R}^{(n-m)\times 1}$ be a randomly generated vector. Let λ_1 be the smallest eigenvalue of A_0, α be a positive constant. Let $A = \begin{pmatrix} A_0 & \mathbf{0}^T \\ \mathbf{0} & (\lambda_1 - \alpha)I \end{pmatrix}$ where $\mathbf{0} \in \mathbb{R}^{m\times(n-m)}$ is the zero matrix, and I is the identity matrix of the proper size. Then we know that $A \in \mathbb{R}^{n\times n}$ is a symmetric sparse matrix, with smallest eigenvalue $\lambda_{\min}(A) = \lambda_1 - \alpha$ and its multiplicity is m. Also, $a = \begin{pmatrix} a_0 \\ \mathbf{0} \end{pmatrix}$ must be orthogonal to the eigenspace of $\lambda_{\min}(A)$. Therefore, if we let $s = 1.1\|(A - \lambda_{\min}(A)I)^{\dagger}a\|$, then $s > \|x(\lambda_{\min}(A))\|$, and so we have a TRS in the hard case (case 2). Moreover, let J be a random permutation of $\{1, 2, ..., n\}$, and we set $A = A(J, J), a = a(J)$, i.e., we permute the rows and columns of A by J, and permute a by J. This does not change the smallest eigenvalue of A, nor its multiplicity. Also, a is still orthogonal to the eigenspace of $\lambda_{\min}(A)$. As a result, the TRS is still in the hard case (case 2), with the multiplicity of smallest eigenvalue m.

From Figure **??**, we can see that the number of matrix-vector multiplications is increasing considerably with the increase of m. This is because, in the implementation of the shift and deflation, we need to calculate m eigenvectors corresponding to the smallest eigenvector in order to make A positive definite and so the hard case (case 2) does not hold. Moreover, since these m eigenvectors used in the shift and deflation are orthogonal, we cannot use the current eigenvector as the starting vector for the next eigenvalue calculation.

Even though it is sensitive to m, the revised RW algorithm is able to solve the TRS in the hard case (case 2) with the desired accuracy (which is 10^{-8}) when $m \leq 20$ in the tests. However, the LSTRS fails when $m > 1$. Precisely, $||(A - \lambda^* I)x^* - a||/||a|| > 1$ or $(\|x^*\| - s)/s > 0.5$ for the LSTRS when it terminates.

Figure 24: Revised RW Algorithm; Varying m; $n = 10000$, Density$= 0.01$; Hard Case (Case 2)



8.8 Results on Unconstrained Optimization Problems

In this section, we compare our algorithm with the MATLAB Large-Scale Unconstrained Optimizer called $fminunc$, which uses the techniques developed in Coleman and Li [**?**, **?**, **?**]. $fminunc$ is also a trust region based method which minimizes the function iteratively. However, in each iteration, it obtains three different solutions to compare —— a 2-D subspace solution, a reflective 2-D subspace solution and a solution on the direction of the gradient. The best among these three solutions are selected to take a step in order to decrease the value of the objective function.

We choose a few functions in [**?**] to perform the test. The algorithm terminates when the first order optimality (the norm of the gradient) is less than the tolerance, which we set to be 10^{-12}. Note that this accuracy is quite high, and because $fminunc$ only gives an approximate solution to the (TRS) in each iteration, this sometimes causes the $fminunc$ fails in achieving the desired accuracy. Both algorithms are trust region based method, where gradient and Hessian are given by a defined function. For some of the problems, we test them with different problem sizes in order to have a more precise results about the two algorithms. Since the cost of time can be affected by many factors (e.g., other running programs, available RAM), all results are obtained by the average of time used to solve the problem 10 times. We also show the norm of the gradient when $fminunc$ can not obtain the desired accuracy (10^{-12}). Results are shown in Table **??**.

First of all, when the norm of the gradient is less than a certain amount (the value shown in the bracket after the cost of time, which depends on different problems), no further improvement can be made by $fminunc$. Precisely, in $fminunc$, the value of the objective function and the gradient remain the same for several iterations and the trust region radius becomes sufficiently small (usually much smaller than the tolerance), then $fminunc$ terminates. On the other hand,

Function	Size	RW iter	RW sec	$fminunc$ iter	$fminunc$ sec	$fminunc$ |gradient|
ARWHEAD	500	9	0.538367	11	0.382102	
ARWHEAD	3000	10	4.888472	11	4.068102	
Broyden Tridiagonal	500	32	10.138823	400	140.324756	3.87e−07
BRYBND	500	26	2.867038	27	1.530577	
BRYBND	2000	31	21.584817	29	18.705026	
DIAGONAL1	500	40	10.249686	159	5.518354	1.25e−06
DIAGONAL2	500	37	14.565753	29	4.932759	
DIAGONAL3	500	13	3.341224	15	3.374957	5.45e−10
DQDRTIC	500	8	0.222984	8	0.192932	
DQDRTIC	5000	11	2.337161	12	2.586480	
GENROSE	500	8	0.413776	7	0.339784	
GENROSE	5000	12	4.537010	11	6.788770	
NONDIA	500	11	1.484020	21	13.210778	
PERTQUA	500	2	0.602417	3	0.231616	6.26e−12
POWER	500	5	0.344326	2	0.132514	
POWER	5000	5	1.059508	2	0.950641	
Raydan1	500	12	5.825982	21	0.823498	8.34e−07
Raydan2	500	20	0.847983	21	0.481980	2.90e−10
SENSORS	500	2	1.439267	2	1.504251	
TRIDIA	500	5	0.389786	5	0.213572	7.12e−09

Table 1: Revised RW Algorithm vs $fminunc$

the revised RW algorithm can still proceed until the desired accuracy is attained. Therefore, the revised RW algorithm can obtain higher accuracy in many problems than $fminunc$.

Secondly, in some problems even though the revised RW algorithm takes less iterations to solve a problem, $fminunc$ is faster in terms of the cost of time. This is because the $fminunc$ only solves an approximate solution in each iteration, while the revised RW algorithm gives a much more accurate estimate. This result is more evident when the $fminunc$ fails to obtain the desired accuracy. For example, in the DIAGONAL1 function, even though the $fminunc$ takes 159 iterations, but most of the iterations actually do not improve the solution at all, and so they cost little time.

Finally, in each iteration, $fminunc$ only solves an approximate solution to the TRS, and so the time spent is much less. However, in the revised RW algorithm, if a step is not taken, (which means everything remains the same for the TRS except the trust region radius), because the warm start is exploited, the gradient or Hessian is not recalculated, and the new TRS is also solved efficiently, and therefore the cost is also quite low.

9 Conclusion

In this thesis we present a survey of the trust region subproblem algorithms and suggest some improvements on the RW algorithm. The well-known optimality conditions for the TRS and the essentially equivalent theorem —— S-lemma are illustrated. After introducing the main developments of the TRS algorithms in the last few decades, we then give detailed review on the MS algorithm, the GLTR algorithm, the SSM algorithm and the Gould-Robinson-Thorne algorithm.

Then, we focus on the RW algorithm and discuss how it can be improved.

Then, simple scaling, a new method which can adjust the trust region radius of the TRS, and how it can be applied to improve the RW algorithm is illustrated. Different choices of the scalar, and some properties of the scaled problem are also introduced. The shift and deflation, a technique used to handle the hard case (case 2) efficiently, is also introduced. It not only solves the hard case (case 2) efficiently by making the matrix A positive definite, but also helps the RW algorithm solving the hard case (case 1) and the easy case by separating the optimal dual variable from the smallest eigenvalue of A. In addition, some other techniques to improve the RW algorithm are also presented, including the rotation modification, the inverse polynomial interpolation and the approximate eigenvalue calculation. We also provide the sensitivity analysis on the TRS, and show that the optimal objective value is stable with respect to the trust region radius in both the easy case and the hard case.

Moreover, we introduce the trust region method, the main part of which is solving a TRS in each iteration. We then propose two modifications to the classic trust region method, i.e., (i) warm start is used when a step is not taken in one iteration; (ii) a different updating scheme for the trust region radius is applied when the hard case (case 2) occurs.

Finally, numerical tests are given to show the performance of the techniques discussed in the thesis. Then we compare the revised RW algorithm with the LSTRS, and show that the revised RW algorithm is more efficient in the hard case (case 2). In particular, the RW algorithm is able to solve the TRS in the hard case (case 2) with multiple smallest eigenvalue, while the LSTRS fails. We also compare the trust region method which applies the revised RW algorithm with the unconstrained optimizer in MATLAB, and show that our algorithm can obtain solutions with higher accuracy than the MATLAB unconstrained optimizer.