AN ELLIPSOID ALGORITHM FOR EQUALITY-CONSTRAINED NONLINEAR PROGRAMS

By

Sharmila Shah

A Thesis Submitted to the Graduate
Faculty of Rensselaer Polytechnic Institute
in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY
Major Subject: Mathematics

Approved by the
Examining Committee:

Dr. Michael Kupferschmid, Thesis Adviser

Dean Joseph G. Ecker, Thesis Adviser

Prof. John Mitchell, Member

Prof. George Habetler, Member

Prof. Albert S. Paulson, Member

Rensselaer Polytechnic Institute
Troy, New York

August 1998
(For graduation December 1998)

UMI Number: 9918402

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acknowledgment

I wish to take this opportunity to express my heartfelt gratitude towards my advisors Dr. Michael Kupferschmid and Dean Joseph Ecker, and the members on my thesis committee, Professors John Mitchell, George Habetler, and Albert Paulson. I also want to thank the faculty and staff of the Department of Mathematics for their help and kind words. I especially wish to thank Dr. Kupferschmid for the countless hours he spent on working with me, and for his words of encouragement and wisdom, and Prof. Mitchell for his help at all the crucial junctures.

This acknowledgment would not be complete without a word of thanks for my friends in this department and Srini Ramaswamy and Srini Rajagopalan from the DSES department. I want to thank all the members of my family in Sangli and Mumbai. I could have never done this without the love and virtual support from Aai, Pallu and Sona. Last but not the least I wish to thank the two most important people in my life, my husband Sachin and my daughter Akanksha.

Abstract

The ellipsoid algorithm is a simple method that yields accurate solutions to convex and many nonconvex nonlinear programming problems. Unfortunately, convergence of the method requires that the feasible set be of full dimension, so it cannot be used with equality constraints.

This thesis presents a variant of the ellipsoid algorithm that solves convex problems having linear equality constraints with or without inequality constraints. The experimental results presented show that the new method is also effective for some problems that are nonconvex or that have nonlinear equality constraints.

CHAPTER 1
Introduction

The nonlinear programming problem is to find a point $\boldsymbol{x}^* \in \Re^n$ such that

$$\begin{aligned} f_i(\boldsymbol{x}^*) &\leq 0 \quad \text{for } i = 1 \ldots m_I \\ f_i(\boldsymbol{x}^*) &= 0 \quad \text{for } i = m_I + 1 \ldots m_I + m_E \\ \text{and} \quad f_0(\boldsymbol{x}^*) &\text{ is } \quad \text{as small as possible,} \end{aligned}$$

where $f_i : \Re^n \to \Re$ for $i = 0 \ldots m_I + m_E$ are continuous real-valued functions. The function f_0 is called the objective function and the other functions are called the constraints. The nonlinear programming problem can be written as

$$\begin{aligned} \min_{\boldsymbol{x} \in \Re^n} \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \leq 0, \quad i = 1 \ldots m_I \\ & f_i(\boldsymbol{x}) = 0, \quad i = m_I + 1 \ldots m_I + m_E. \end{aligned}$$

We will assume that all the functions are differentiable and that the feasible set S is nonempty.

$$S = \{\boldsymbol{x} \in \Re^n | f_i(\boldsymbol{x}) \leq 0 \text{ for } i = 1 \ldots m_I, \ f_i(\boldsymbol{x}) = 0 \text{ for } i = m_I + 1 \ldots m_I + m_E\} \neq \emptyset$$

A nonlinear programming problem is convex if $m_E = 0$ and the f_i are convex functions, or if $m_E > 0$, the f_i are linear for $i = m_I + 1 \ldots m_I + m_E$ and the other functions are convex.

1.1 Finding The Sphinx in the Sahara

The ellipsoid algorithm was first introduced to solve convex programming problems having only inequality constraints. The method has its roots in convex nondifferentiable optimization (subgradient, space dilation methods, methods of central sections) as well as in studies on the computational complexity of convex programming problems. N. Z. Shor [16] gave the first explicit statement of the ellipsoid algorithm. Yudin and Nemirovskii [18] discussed a similar approach to convex problems in studying the complexity of convex extremal problems. L. G. Khachian [13] used it to prove the existence of polynomial algorithms for linear programming problems. Bland, Goldfarb and Todd give the history of the ellipsoid method in their survey paper [2]. Various authors have shown the adaptability of the ellipsoid method to linear and combinatorial optimization; for more information see [9], [2]. Ecker and Kupferschmid [4], [5] have provided computational evidence that the ellipsoid method is effective for smooth nonlinear optimization.

In this dissertation I am interested in the ellipsoid algorithm for nonlinear programming problems. The classical ellipsoid algorithm can be used only for convex programming problems without equality constraints, i.e., problems of the type

$$\begin{array}{rrcl} \min\limits_{x \in \Re^n} & f_0(x) & & \\ \text{subject to} & f_i(x) & \leq & 0, \quad i = 1 \ldots m_I. \end{array}$$

To get a rough idea of how the ellipsoid algorithm works, consider the following method of locating The Sphinx in the Sahara desert [9]. Put a fence around the Sahara, and split it into two parts; check which part contains The Sphinx, fence-in the part containing The Sphinx, and continue. After a finite number of steps, either the fenced-in zone will be so small that The Sphinx is seen, or we realize that the fenced-in zone is so small that it cannot contain The Sphinx, i.e., we never had The Sphinx to start with. The ellipsoid algorithm is a similar space confinement method. The rest of this chapter gives a description of the ellipsoid algorithm and the conditions under which it works.

1.2 The Ellipsoid Algorithm

Starting with given bounds U and L containing an optimal point x^* for a convex programming problem, the ellipsoid algorithm generates a sequence of ellipsoids E_k, each guaranteed to contain $\boldsymbol{x}^*$, with the property that their volumes shrink to zero as the terms of a geometric progression. The convergence rate of an ellipsoid algorithm depends on the rate at which the volumes of the successive ellipsoids shrink to zero.

From the upper and lower bounds $\boldsymbol{U}$ and $\boldsymbol{L}$, we can calculate $\boldsymbol{Q}_0$, the inverse matrix for the starting ellipsoid E_0 and its center $\boldsymbol{x}^0$ in the following way.

$$\boldsymbol{x}^0 = \frac{(\boldsymbol{U}+\boldsymbol{L})}{2}$$

$$\text{and} \quad \boldsymbol{Q}_0 = \frac{n}{4}\begin{pmatrix} (U_1-L_1)^2 & 0 & \dots & 0 \\ 0 & (U_2-L_2)^2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & (U_n-L_n)^2 \end{pmatrix}$$

The ellipsoid is given by $E_0 = \{\boldsymbol{x} \in \Re^n \mid (\boldsymbol{x}-\boldsymbol{x}^0)^T \boldsymbol{Q}_0^{-1}(\boldsymbol{x}-\boldsymbol{x}^0) \leq 1\}$, where $\boldsymbol{Q}_0$ is positive definite and symmetric. At the first iteration, we have the ellipsoid that contains the optimal point $\boldsymbol{x}^*$ and a part or all of the feasible set S. Now, to split this ellipsoid in two parts, we look at two scenarios.

- The first one is when the current center is not feasible. Here we find the supporting hyperplane to a violated constraint at the center $\boldsymbol{x}^k$ and use that hyperplane to split the ellipsoid into two halves. This hyperplane is called the cutting hyperplane.

- The second scenario is when the center is feasible because there are no violated constraints. In this case we use the supporting hyperplane to the contour of the objective function at the center as the cutting hyperplane.

The supporting hyperplane is given by $H_k = \{\boldsymbol{x} \mid -\nabla f_i(\boldsymbol{x}^k)^T(\boldsymbol{x}-\boldsymbol{x}^k) = 0\}$, where $f_i(\boldsymbol{x})$ is either the violated constraint as in the first scenario, $i = 1 \dots m_I$, or the objective function f_0 as in the second scenario, and $\nabla f_i(\boldsymbol{x}^k) \neq 0$. The optimal point lies in one of the half-spaces of the cutting hyperplane.

Next, we need to take a step in the direction $\boldsymbol{d}$ such that we go as far away from the current center $\boldsymbol{x}^k$ as possible, in the appropriate half space, and the next point we get by taking the step is still in the ellipsoid $\boldsymbol{E}_k$. In finding the supporting hyperplane, we use the normalized gradient $\boldsymbol{g}$ of the appropriate function.

$$\boldsymbol{g} = \frac{\nabla f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|_2}$$

We can find the direction $\boldsymbol{d}$ by solving the following problem.

$$\begin{array}{ll} \min\limits_{\boldsymbol{d} \in \Re^n} & \boldsymbol{g}^T \boldsymbol{d} \\ \text{subject to} & \boldsymbol{d}^T \boldsymbol{Q}_k^{-1} \boldsymbol{d} \leq 1 \end{array}$$

The above problem can be solved to find direction $\boldsymbol{d}$ by using the Karush-Kuhn-Tucker (KKT) conditions [1]. The KKT conditions give us

$$\begin{aligned} \boldsymbol{g} + 2u\boldsymbol{Q}_k^{-1}\boldsymbol{d} &= 0 && (1.1) \\ u(\boldsymbol{d}^T\boldsymbol{Q}_k^{-1}\boldsymbol{d} - 1) &= 0 && (1.2) \\ u &\geq 0. && (1.3) \\ \text{From (1.1) we see that } 2u\boldsymbol{Q}_k^{-1}\boldsymbol{d} &= -\boldsymbol{g} \\ \text{so } 2u\boldsymbol{d} &= -\boldsymbol{Q}_k\boldsymbol{g}. && (1.4) \end{aligned}$$

It must be that $u > 0$, because if $u = 0$, we would have from (1.4) $\boldsymbol{Q}_k\boldsymbol{g} = \boldsymbol{0}$. This is impossible, because $\boldsymbol{g}$ is not zero and $\boldsymbol{Q}_k$ is a positive definite matrix. Thus by (1.3), (1.4) and (1.2), we get

$$\begin{aligned} u &> 0 \\ \boldsymbol{d} &= -\frac{1}{2u}\boldsymbol{Q}_k\boldsymbol{g} \\ \boldsymbol{d}^T\boldsymbol{Q}_k^{-1}\boldsymbol{d} - 1 &= 0. \end{aligned}$$

Substituting for $\boldsymbol{d}$ in the last equality,

$$\begin{aligned} (-\frac{1}{2u}\boldsymbol{Q}_k\boldsymbol{g})^T\boldsymbol{Q}_k^{-1}(-\frac{1}{2u}\boldsymbol{Q}_k\boldsymbol{g}) &= 1 \\ \frac{1}{4u^2}\boldsymbol{g}^T\boldsymbol{Q}_k\boldsymbol{Q}_k^{-1}\boldsymbol{Q}_k\boldsymbol{g} &= 1 \end{aligned}$$

$$\begin{aligned}\frac{1}{4u^2}g^TQ_kg &= 1\\ \frac{1}{4u^2} &= \frac{1}{g^TQ_kg}\\ \frac{1}{2u} &= \frac{1}{\sqrt{g^TQ_kg}}\end{aligned}$$

The positive square root gives the correct direction for $\boldsymbol{d}$. Thus using the KKT conditions we get

$$\boldsymbol{d} = -\frac{Q_kg}{\sqrt{g^TQ_kg}}.$$

This is the direction that minimizes $g^T\boldsymbol{d}$ over the ellipsoid E_k. The direction $\boldsymbol{d}$ lies in the half-space containing the optimal point x^*. The tangent hyperplane to the ellipsoid at the point $x^k + \boldsymbol{d}$ on the boundary of the ellipsoid is parallel to the cutting hyperplane H_k used to generate $\boldsymbol{d}$,

$$H_k = \{x \in \Re^n \mid g^T(x - x^k) = 0\}. \tag{1.5}$$

Translating H_k parallel to itself until it is tangent to an ellipsoid E_k at a point $y = x^k + \boldsymbol{d}$ where $E_k = \{x \in \Re^n \mid (x - x^k)^TQ_k^{-1}(x - x^k) = 1\}$ yields [1]

$$\begin{aligned}(Q_k^{-1}(y - x^k))^T(x - y) &= 0.\\ (Q_k^{-1}(x^k + d - x^k))^T(x - x^k - d) &= 0\\ (Q_k^{-1}d)^T(x - x^k - d) &= 0\\ \text{But } d &= -\frac{Q_kg}{\sqrt{g^TQ_kg}}\\ \text{so } \left(Q_k^{-1}\left(-\frac{Q_kg}{\sqrt{g^TQ_kg}}\right)\right)^T\left(x - x^k - \left(-\frac{Q_kg}{\sqrt{g^TQ_kg}}\right)\right) &= 0\\ (Q_k^{-1}Q_kg)^T\left(x - x^k + \left(\frac{Q_kg}{\sqrt{g^TQ_kg}}\right)\right) &= 0\\ g^T(x - x^k) + \left(\frac{g^TQ_kg}{\sqrt{g^TQ_kg}}\right) &= 0\\ g^T(x - x^k) + \sqrt{g^TQ_kg} &= 0.\end{aligned}$$

Thus the equation for the tangent hyperplane to the ellipsoid at a point $\boldsymbol{y} = \boldsymbol{x}^k + \boldsymbol{d}$ is given by

$$\boldsymbol{g}^T(\boldsymbol{x} - \boldsymbol{x}^k) = -\sqrt{\boldsymbol{g}^T \boldsymbol{Q}_k \boldsymbol{g}}. \tag{1.6}$$

Next, we will find the new ellipsoid E_{k+1} that contains the desired half of the ellipsoid E_k. We know that every convex body is contained in a unique ellipsoid of minimal volume and contains a unique ellipsoid of maximal volume. These two results have been discovered by several mathematicians independently, but the first result is attributed to to K. Löwner [9]. Fritz John [12] has proved the above result in a more general context. Let us call the desired new ellipsoid the Löwner-John ellipsoid [9]. For convex bodies that are particular ellipsoidal sections, the formulas for Löwner-John ellipsoids are known. For ellipsoid E_k with center $\boldsymbol{x}^k$ and cutting hyperplane H_k, let us set

$$E_k' = E_k \bigcap \{\boldsymbol{x} \in \Re^n | \boldsymbol{g}^T(\boldsymbol{x} - \boldsymbol{x}^k) \leq 0\}.$$

E_k' is the half-ellipsoid obtained by cutting E_k through its center $\boldsymbol{x}^k$ using the hyperplane H_k. The Löwner-John ellipsoid containing E_k' is the ellipsoid given by $E_{k+1} = \{\boldsymbol{x} \in \Re^n \mid (\boldsymbol{x} - \boldsymbol{x}^{k+1})^T \boldsymbol{Q}_{k+1}^{-1}(\boldsymbol{x} - \boldsymbol{x}^{k+1}) \leq 1\}$ with center $\boldsymbol{x}^{k+1}$ where [9] [14]

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \frac{1}{n+1}\boldsymbol{d} \\ \boldsymbol{Q}_{k+1} &= \frac{n^2}{n^2-1}\left(\boldsymbol{Q}_k - \frac{2}{n+1}\boldsymbol{d}\boldsymbol{d}^T\right). \end{aligned}$$

Now we are ready to describe the algorithm.

We are given NLP :

$$\begin{aligned} \min_{\boldsymbol{x} \in \Re^n} \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \leq 0, \quad i = 1 \ldots m_I, \end{aligned}$$

starting bounds $\boldsymbol{U}$ and $\boldsymbol{L}$, and a convergence tolerance T.

0. Initialize

$$\boldsymbol{x}^0 = \frac{(\boldsymbol{U}+\boldsymbol{L})}{2}$$

$$\text{and} \quad \boldsymbol{Q}_0 = \frac{n}{4}\begin{pmatrix} (U_1-L_1)^2 & 0 & \dots & 0 \\ 0 & (U_2-L_2)^2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & (U_n-L_n)^2 \end{pmatrix}$$

1. Select a violated inequality constraint or the objective.

$$\text{if } f_I(\boldsymbol{x}^k) > 0 \text{ for some } I, \text{ set } i = I$$
$$\text{or if } f_i(\boldsymbol{x}^k) \leq 0 \text{ for all } i = 1 \dots m_I, \text{ then set } i = 0$$

2. Find the unit normal vector to the cutting hyperplane.

$$\boldsymbol{g} = \frac{\nabla f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|}$$

3. Find the direction in which to move $\boldsymbol{x}$.

$$\boldsymbol{d} = -\frac{\boldsymbol{Q}_k\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T\boldsymbol{Q}_k\boldsymbol{g}}}$$

4. Update $\boldsymbol{x}$ and $\boldsymbol{Q}$.

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \frac{1}{n+1}\boldsymbol{d}$$
$$\boldsymbol{Q}_{k+1} = \frac{n^2}{n^2-1}\left(\boldsymbol{Q}_k - \frac{2}{n+1}\boldsymbol{d}\boldsymbol{d}^T\right)$$

5. Check for convergence.

if $\|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\| < T$, STOP ; $\boldsymbol{x}^{k+1}$ is close enough to the minimizing point.

6. Repeat

increase k by 1.

GO TO 1.

1.3 Theoretical Conditions for Convergence

Jean-Louis Goffin [8] and later Hans-Jacob Luthi [15] have given proofs of convergence for the ellipsoid algorithm. The ellipsoid algorithm can be described as a variable metric subgradient optimization method, where the matrix is updated at every iteration by a rank-one matrix. Goffin shows that the ellipsoid method converges at a geometric rate that depends only on the dimension of the space but not on the actual function to be minimized. The convergence proof requires that the objective function and the constraints be convex functions. It is also necessary that the feasible set is full-dimensional, which means that for a problem where $x \in \Re^n$ the feasible set has dimension n. In the rank-one matrix update of the ellipsoid method, the volumes of the ellipsoids are reduced at each iteration [9] according to $V[E_{k+1}] = c_n V[E_k]$ or $V[E_k] = (c_n)^k V[E_0]$, where

$$c_n = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2} < 1.$$

The volumes $V[E_k]$ decrease as a geometric sequence whose ratio depends only upon the dimension of the space.

Problems with equality constraints cannot be solved by the ellipsoid method for the following two reasons.

1. For linear equality constraints, the dimension of the feasible set is less than dimension of the space n. The feasible set is a flat in higher-dimensional space.

2. For nonlinear constraints, the feasible set may not be full dimensional and the problem cannot be convex.

1.4 Behavior in Practice

The ellipsoid algorithm is robust for solving inequality-constrained problems that are convex. It is linear in convergence and hence slower than some higher-order methods like the generalized reduced gradient method, but much more reliable and capable of finding precise solutions [5]. The ellipsoid algorithm is very simple and thus simple for computer implementation as well. Even for nonconvex problems the ellipsoid algorithm often finds global minima and avoids local minima. The ellipsoid algorithm always fails for problems that do not have a full dimensional feasible set. These properties make the ellipsoid algorithm a very good candidate for modification so it can be used for problems with equality constraints.

1.5 Equality Constraints and A New Approach

It would be desirable to have an ellipsoid algorithm for solving equality-constrained problems. I will first consider problems with linear equality constraints. These make the feasible region a flat in the higher-dimensional space but as discussed in §1.3 the ellipsoid algorithm is guaranteed to converge only for a convex and full-dimensional feasible region. Then I will take up problems that have nonlinear equality constraints, which are more difficult because nonlinear equality constraints make the feasible region nonconvex.

My approach to solving equality-constrained problems with the ellipsoid algorithm will be to find a direction $\boldsymbol{d}$ that lies on the flat F of the equality constraints. The ellipsoid algorithm updates can then be used. Here we want to find a new ellipsoid that will always contain the optimal point, which lies on the flat defined by the equality constraints. How I accomplish that is discussed in the next chapter.

CHAPTER 2
The New Algorithm

Recall the nonlinear programming problem with inequality and equality constraints,

$$\begin{array}{rrcl} \min_{\boldsymbol{x}\in\Re^n} & f_0(\boldsymbol{x}) & & \\ \text{subject to} & f_i(\boldsymbol{x}) & \leq & 0, \;\; i = 1\ldots m_I \\ & f_i(\boldsymbol{x}) & = & 0, \;\; i = m_I + 1..m_I + m_E. \end{array}$$

Here $f_i : \Re^n \longrightarrow \Re$ for $i = 0 \ldots m_I + m_E$ are continuous, differentiable functions. Also recall that for ensuring the convergence of the original ellipsoid algorithm (EA), the feasible set $S' = \{\boldsymbol{x} \in \Re^n | f_i(\boldsymbol{x}) \leq 0, i = 1\ldots m_I\}$ must be a convex set and f_0 must be a convex function. For convenience we will make a stronger assumption that all the functions f_0 and f_i for $i = 1 \ldots m_I$ are convex. As described in §1.3, the classical ellipsoid algorithm (EA) is not suited for this problem because the equality constraints cause the feasible set to be of lower dimension than n. Suppose, however, that we apply EA and ignore the equality constraints. If this leads to a sequence of EA iterates $\boldsymbol{x}^k$ each of which, by some remarkable coincidence, happens also to satisfy the equalities, then it would be possible to use EA to solve the above problem. The feasible set S for the above problem can be described as the intersection of the surface F of equality constraints with the feasible set S' for the inequality constraints. As for EA we assume S' is a full-dimensional set in $\Re^n$. Then we have

$$F = \{\boldsymbol{x} \in \Re^n | f_i(\boldsymbol{x}) = 0, i = m_I + 1 \ldots m_I + m_E\} \quad \text{and} \quad S = S' \cap F.$$

2.1 A Two-Dimensional Example

To see how it might be possible to arrange for each $\boldsymbol{x}^k$ to fall on the surface F, consider the case where the equality constraints are linear. Here we can write the equality constraints $f_i(\boldsymbol{x}) = 0$ as $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ or $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A}$ is an $m_E \times n$

matrix with rank m_E and $m_E < n$. The linear equality constraints define the flat $F = \{x \in \Re^n | Ax = b\}$. If we begin with $x^0 \in F$, can we choose each direction vector d_F such that $x^{k+1} = x^k + \frac{1}{n+1} d_F \in F$? Let us consider the following example.

$$\min_{x \in \Re^2} \; 3x_1^2 + x_2^2$$
$$\text{subject to} \quad x_1 + x_2 - 1 \;=\; 0$$

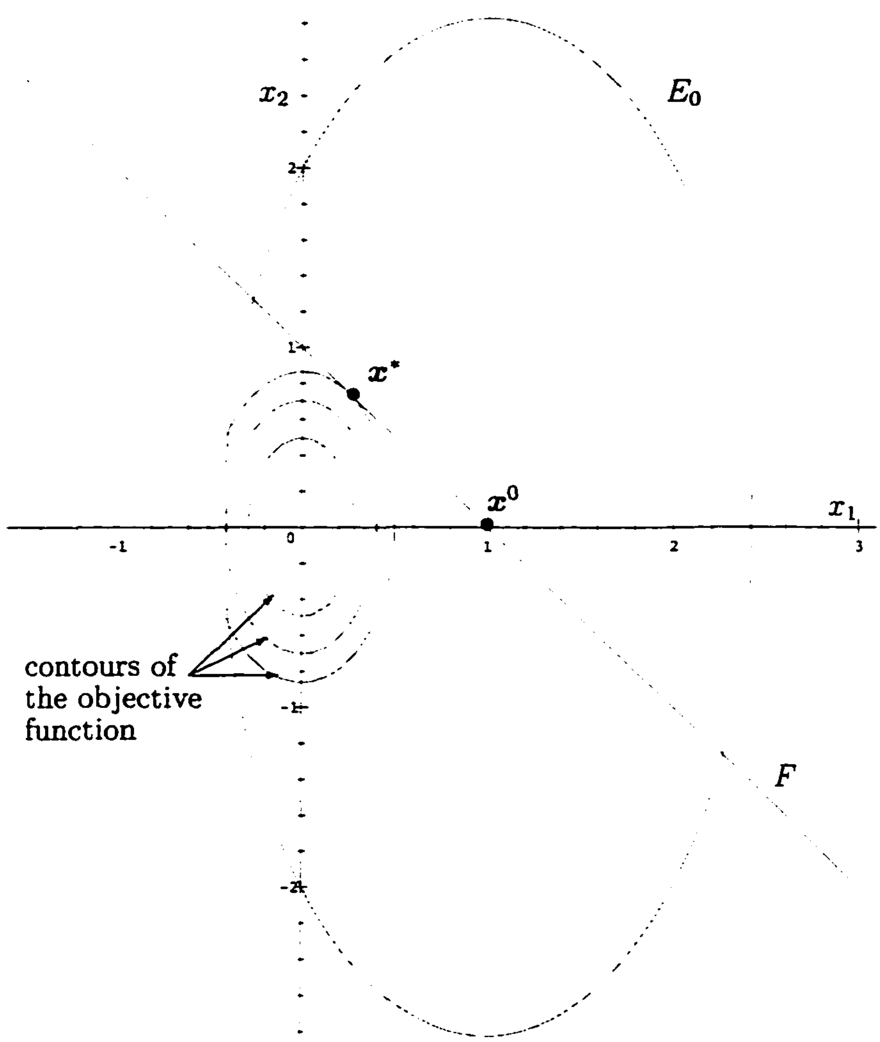


Figure 2.1: The Starting Ellipsoid

We can solve this problem analytically to see that the unconstrained optimal point is $[0, 0]^T$ and the constrained optimal point is $[\frac{1}{4}, \frac{3}{4}]^T$ (see figure 2.1).

The flat of equalities is $F = \{\boldsymbol{x} | x_1 + x_2 - 1 = 0\}$.

$$\text{We will use } \boldsymbol{x}^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } Q_0 = \begin{bmatrix} 2 & 0 \\ 0 & 8 \end{bmatrix}$$

$$\text{so that } E_0 = \{\boldsymbol{x} \in \Re^2 | (\boldsymbol{x} - \boldsymbol{x}^0)^T \boldsymbol{Q}_0^{-1} (\boldsymbol{x} - \boldsymbol{x}^0) \leq 1\}.$$

For this two-dimensional problem I will use the projection of the normalized gradient $\boldsymbol{g}$ on the flat of equalities as the desired direction $\boldsymbol{d}_F$. Later I will generalize this for an n dimensional problem using a different approach.

This problem has no inequality constraints, so we will look at the contour of the objective function at $\boldsymbol{x}^0$ to find $\boldsymbol{g}$. Referring to the algorithm statement in §1.2, we can see that $\boldsymbol{g} = [1, 0]^T$. If we project $\boldsymbol{x}^0 - \boldsymbol{g}$ onto the flat F we get $\boldsymbol{w}_F = [0.293, 0.707]^T$. For the purpose of this discussion, the projections are done geometrically. As we can see in figure 2.2, $\boldsymbol{w}_F$ is not a point in the boundary of the starting ellipse E_0. For the update formulas for $\boldsymbol{x}$ and $\boldsymbol{Q}$ in EA, it is necessary that $\boldsymbol{x}^k + \boldsymbol{d}$ is a point in the boundary of the current ellipse $\boldsymbol{E}^k$. Consider the point in the boundary of the ellipse we can get by moving from $\boldsymbol{x}^0$ in the direction $\boldsymbol{d}_F$. We can call this point $\boldsymbol{x}^0 + \mu\boldsymbol{d}_F$, where μ is the appropriate scaling factor for $\boldsymbol{d}_F$. Using this $\mu\boldsymbol{d}_F$, we can find the next ellipse E_1 and its center. For this problem,

$$\begin{aligned} \mu\boldsymbol{d}_F &= \begin{bmatrix} -1.265 \\ 1.265 \end{bmatrix} \\ \boldsymbol{x}^1 = \boldsymbol{x}^0 + \frac{1}{n+1}\mu\boldsymbol{d}_F &= \begin{bmatrix} 0.578 \\ 0.422 \end{bmatrix} \\ \boldsymbol{Q}_1 &= \begin{bmatrix} 1.24 & 1.42 \\ 1.42 & 9.24 \end{bmatrix}. \end{aligned}$$

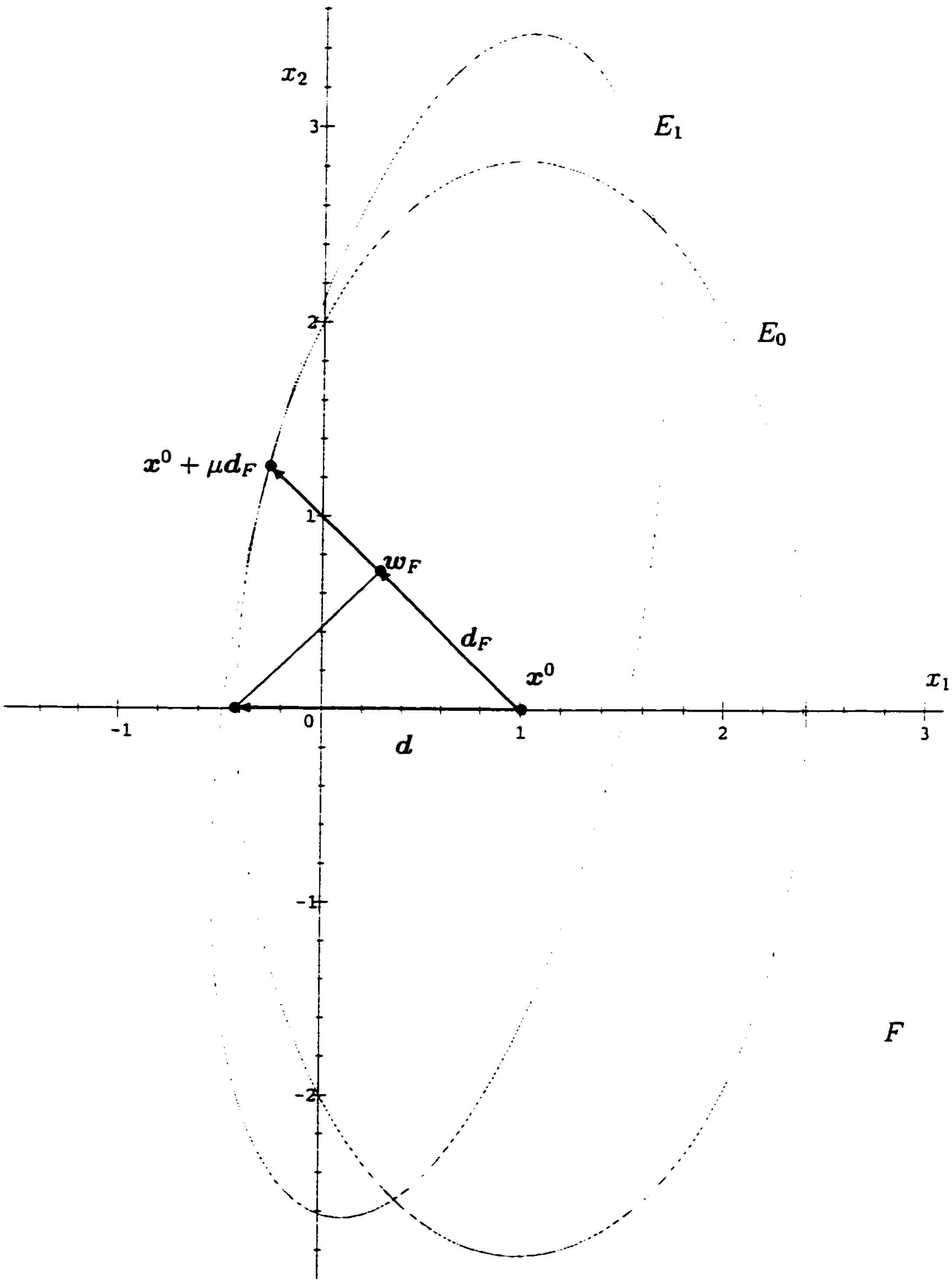


Figure 2.2: The First Two Ellipsoids

Let us find the next $\boldsymbol{g}$ at $\boldsymbol{x}^1$ and do another EA iteration (not shown in figure 2.2). Again if we project $\boldsymbol{x}^1 - \boldsymbol{g}$ on the flat F, we get the next $\boldsymbol{w}_F$. To get the next ellipsoid, we begin by finding the hyperplane that is tangent to the current ellipsoid at $\boldsymbol{x}^1 + \mu \boldsymbol{d}_F$. Recall that this point is in the boundary of the ellipsoid. We then find the hyperplane that goes through the center $\boldsymbol{x}^1$ and that is parallel to this other hyperplane. The next ellipsoid will be the smallest ellipsoid that goes through $\boldsymbol{x}^1 + \mu \boldsymbol{d}_F$ and also the intersection of the hyperplane through the center with E_1. Using the update formulas in EA will give us the new ellipsoid and its center. Continuing this process will take us to the optimal point. The above problem gives a simple picture of what we can do with a two dimensional problem, but we need a more general approach to solve higher dimensional problems.

2.2 Finding $\boldsymbol{d}$ in General

For problems with linear equality constraints, we can use an approach similar to the one used in Chapter 1 to find $\boldsymbol{d}$. We first ignore the equality constraints and find the appropriate $\boldsymbol{g}$. Then we want to find the direction $\boldsymbol{d}$ that will lead us to a next ellipsoid whose center also satisfies the equality constraints. Similar to the case of inequality constrained problems, we can state this problem as minimize $\boldsymbol{g}^T\boldsymbol{x}$ over the intersection of the ellipsoid with the equality constraints. The minimizing point is $\boldsymbol{x} = \boldsymbol{x}^k + \boldsymbol{d}$, where $\boldsymbol{x}^k$ is the current center. So we can rewrite the problem of finding $\boldsymbol{d}$ as

$$
\begin{array}{rl}
\min\limits_{d \in \Re^n} & \boldsymbol{g}^T(\boldsymbol{x}^k + \boldsymbol{d}) \\
\text{subject to} & ((\boldsymbol{x}^k + \boldsymbol{d}) - \boldsymbol{x}^k)^T Q_k^{-1}((\boldsymbol{x}^k + \boldsymbol{d}) - \boldsymbol{x}^k) \leq 1 \\
& \boldsymbol{A}(\boldsymbol{x}^k + \boldsymbol{d}) = \boldsymbol{b}.
\end{array}
$$

If $\boldsymbol{x}^k$ satisfies the equalities $\boldsymbol{A}\boldsymbol{x}^k = \boldsymbol{b}$, then ignoring the constant $\boldsymbol{g}^T\boldsymbol{x}^k$ in the objective, this problem is equivalent to,

$$
\begin{array}{rl}
\min\limits_{d \in \Re^n} & \boldsymbol{g}^T\boldsymbol{d} \\
\text{subject to} & \boldsymbol{d}^T Q_k^{-1} \boldsymbol{d} \leq 1 \\
& \boldsymbol{A}\boldsymbol{d} = 0.
\end{array}
$$

In the next section we will consider the case where $\boldsymbol{x}^k$ does not satisfy the equality constraints. Now we can find the direction $\boldsymbol{d}$ analytically from the KKT conditions as before. The KKT conditions for the above problem give us

$$\boldsymbol{g} + 2u\boldsymbol{Q}^{-1}\boldsymbol{d} + \boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{0} \tag{2.1}$$

$$u(\boldsymbol{d}^T\boldsymbol{Q}_k^{-1}\boldsymbol{d} - 1) = 0 \tag{2.2}$$

$$u \geq 0 \tag{2.3}$$

$$\boldsymbol{A}\boldsymbol{d} = \boldsymbol{0} \tag{2.4}$$

where $\boldsymbol{y}$ is a vector that is unconstrained in sign. In order to find the optimum $\boldsymbol{d}$, we need to consider two cases, namely $u = 0$ and $u > 0$.

If $u = 0$, then (2.1) becomes $\boldsymbol{g} + \boldsymbol{A}^T\boldsymbol{y} = \boldsymbol{0}$. This implies that $\boldsymbol{g} = -\boldsymbol{A}^T\boldsymbol{y}$, which means $\boldsymbol{g}$ is a normal to the flat of equalities and $\boldsymbol{d} = \boldsymbol{0}$, the trivial solution. Geometrically this means that the cutting hyperplane contains the flat of equalities F. If $\boldsymbol{g}$ is the normalized gradient of a violated inequality constraint, then the problem is infeasible since $S' \cap F = \emptyset$. If $\boldsymbol{g}$ is the normalized gradient of the objective function, then the current center is the optimal point. Thus, if $u = 0$ in (2.1)-(2.4) then either the original program NLP is infeasible or $\boldsymbol{x}^k$ is the optimal point.

If NLP is feasible and $\boldsymbol{x}^k$ is not the optimal point, then $u > 0$. Multiplying equation (2.1) by $\boldsymbol{AQ}$ we get

$$\begin{aligned} \boldsymbol{AQg} + 2u\boldsymbol{AQQ}^{-1}\boldsymbol{d} + \boldsymbol{AQA}^T\boldsymbol{y} &= 0 \\ \boldsymbol{AQg} + 2u\boldsymbol{Ad} + \boldsymbol{AQA}^T\boldsymbol{y} &= 0. \end{aligned}$$

But from (2.4) we have $\boldsymbol{Ad} = \boldsymbol{0}$. Thus,

$$\begin{aligned} \boldsymbol{AQg} + \boldsymbol{AQA}^T\boldsymbol{y} &= \boldsymbol{0} \\ \boldsymbol{AQA}^T\boldsymbol{y} &= -\boldsymbol{AQg} \\ \boldsymbol{y} &= -(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQg} \end{aligned}$$

The matrix $\boldsymbol{AQA}^T$ is invertible because $\boldsymbol{Q}$ is symmetric positive definite and $\boldsymbol{A}$ has full rank m_E. Substituting this $\boldsymbol{y}$ back into (2.1) and then multiplying by $\boldsymbol{Q}$ and rearranging gives

$$
\begin{aligned}
2ud &= -Qg + QA^T(AQA^T)^{-1}AQg \\
d &= -\frac{1}{2u}(Qg - QA^T(AQA^T)^{-1}AQg) \\
d &= -\frac{1}{2u}(Q - QA^T(AQA^T)^{-1}AQ)g
\end{aligned} \tag{2.5}
$$

To find d, we need to know u. Since $u > 0$, (2.2) gives us $d^TQ^{-1}d = 1$. Substituting the d from (2.5) in $d^TQ^{-1}d = 1$, we can find $2u$.

$$
\begin{aligned}
\frac{1}{4u^2}g^T(Q - QA^T(AQA^T)^{-1}AQ)Q^{-1}(Q - QA^T(AQA^T)^{-1}AQ)g &= 1 \\
\frac{1}{4u^2}g^T(I - QA^T(AQA^T)^{-1}A)(Q - QA^T(AQA^T)^{-1}AQ)g &= 1 \quad (2.6) \\
g^T(Q - QA^T(AQA^T)^{-1}AQ - QA^T(AQA^T)^{-1}AQ & \\
+ QA^T\underbrace{(AQA^T)^{-1}AQA^T}_{I}(AQA^T)^{-1}AQ)\, g &= 4u^2 \\
g^T(Q - QA^T(AQA^T)^{-1}AQ - QA^T(AQA^T)^{-1}AQ & \\
+QA^T(AQA^T)^{-1}AQ)g &= 4u^2 \\
g^T\,(Q - QA^T(AQA^T)^{-1}AQ)\,g &= 4u^2 \quad (2.7) \\
\sqrt{g^T\,(Q - QA^T(AQA^T)^{-1}AQ)\,g} &= 2u
\end{aligned}
$$

This ensures that $\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}$ is nonzero. Thus the direction d that leads us to the next ellipsoid is given by

$$
d = -\frac{(Q - QA^T(AQA^T)^{-1}AQ)g}{\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}}.
$$

2.3 Starting From Off the Flat

In the case of linear equality constraints, at every iteration the current ellipsoid center is always on the flat of equality constraints because of the KKT conditions discussed analytically in the previous section. In reality, roundoff errors invariably cause the new center to be slightly off the flat F. Also in the case of nonlinear equalities, the current center may not be on the flat given by linearization of the equalities. If the ellipsoid center x^k at any iteration is not on the flat F of equalities, then we cannot use the method discussed above to find the new direction. In fact, doing so makes the method numerically unstable. This problem is fixed by projecting each $\boldsymbol{x}^k$ onto the flat of equalities. In this section we find the required projection of $\boldsymbol{x}^k$ on the flat F. We call this projection $\boldsymbol{x}_F^k$.

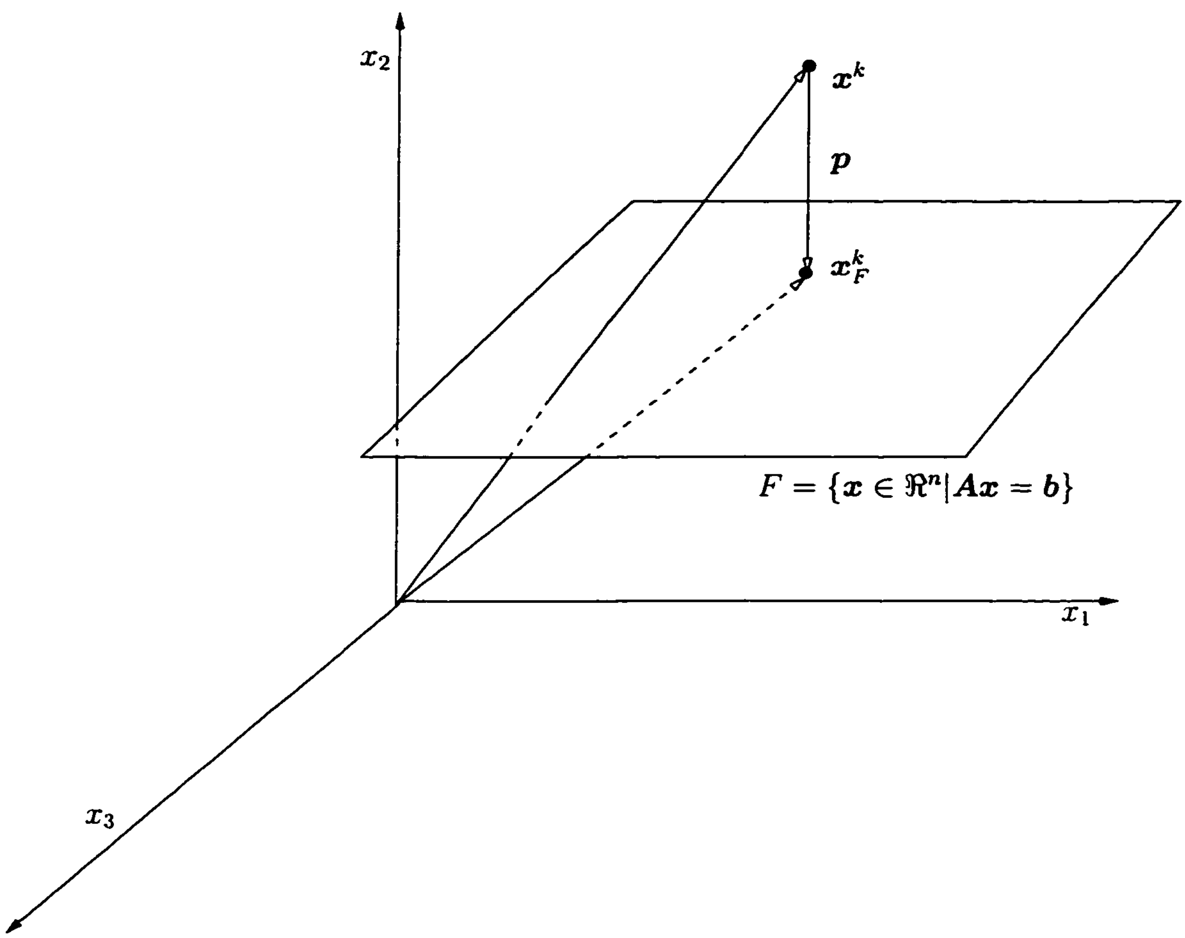


Figure 2.3: Projecting Onto the Flat

Recall that $\boldsymbol{A}$ is an $m_E \times n$ matrix with rank m_E where $m_E < n$. As shown in Figure 2.3 above, $\boldsymbol{x}_F^k = \boldsymbol{x}^k + \boldsymbol{p}$, where $\boldsymbol{p}$ is the normal to F from $\boldsymbol{x}^k$. In geometrical terms $\boldsymbol{x}_F^k$ is the foot of the perpendicular from $\boldsymbol{x}^k$ to F. The vector $\boldsymbol{p}$ is normal to F, so it is orthogonal to any vector lying in F.

Theorem: The normal vector $\boldsymbol{p}$ can be written as a linear combination of the rows of $\boldsymbol{A}$.

Proof: The null space and the row space of a matrix $\boldsymbol{A}$ are orthogonal complements of each other [17]. Thus any vector orthogonal to a vector $\boldsymbol{v}$ in the null space of the matrix $\boldsymbol{A}$ must belong to the row space of $\boldsymbol{A}$. We can write a vector in the row space of $\boldsymbol{A}$ as a linear combination of rows of $\boldsymbol{A}$. So we can write $\boldsymbol{p}$ in the following way.

$$\boldsymbol{p} = A_{1\bullet}\alpha_1 + A_{2\bullet}\alpha_2 + \cdots + A_{m_e\bullet}\alpha_{m_e}$$

where $A_{i\bullet}$ is the i^{th} row of $\boldsymbol{A}$. In other words, $\boldsymbol{p} = \boldsymbol{A}^T\boldsymbol{\alpha}$ where $\boldsymbol{\alpha}$ is a vector of dimension m_E. ∎

Using all the facts we know so far, $\boldsymbol{x}_F^k = \boldsymbol{x}^k + \boldsymbol{A}^T\boldsymbol{\alpha}$. We know $\boldsymbol{x}^k$ and $\boldsymbol{A}$, so to find $\boldsymbol{x}_F^k$, we need to find the vector $\boldsymbol{\alpha}$. We have $\boldsymbol{A}\boldsymbol{x}_F^k = \boldsymbol{b}$, so $\boldsymbol{A}(\boldsymbol{x}^k + \boldsymbol{p}) = \boldsymbol{b}$, which gives us $\boldsymbol{A}(\boldsymbol{x}^k + \boldsymbol{A}^T\boldsymbol{\alpha}) = \boldsymbol{b}$. Simplifying this equation, we get

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x}^k + \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} &= \boldsymbol{b} \\ \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} &= \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k. \end{aligned}$$

We want to avoid finding the inverse of any matrix, so we solve the above set of linear equations to get $\boldsymbol{\alpha}$. Then $\boldsymbol{x}_F^k = \boldsymbol{x}^k + \boldsymbol{A}^T\boldsymbol{\alpha}$ is the required projection of $\boldsymbol{x}^k$ onto the flat F. This $\boldsymbol{x}_F^k$ can be used at each iteration to find the new $\boldsymbol{d}$.

2.4 Nonlinear Constraints

When any of the equality constraints are nonlinear, the algorithm does not necessarily converge. As discussed earlier, the nonlinear equality constraints cause NLP to be nonconvex and the feasible set to be of lower dimension than n. However as discussed further in Chapter 4, the following approach is often successful in practice. At every iteration, we linearize the equality constraints around the current center using a first-order Taylor series approximation. The constraints are continuous and differentiable functions, hence we can use the first two terms in the Taylor series approximation. Using f_i where $i = m_I + 1 \ldots m_I + m_E$, we find $F = \{\boldsymbol{x} \in \Re^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$ in the following way. At the current center $\boldsymbol{x}^k$ we can linearize the nonlinear constraint f_i using

$$\begin{aligned} f_i(\boldsymbol{x}^k) + (\nabla f_i(\boldsymbol{x}^k))^T(\boldsymbol{x} - \boldsymbol{x}^k) &= 0 \\ \text{i. e.,} \quad (\nabla f_i(\boldsymbol{x}^k))^T\boldsymbol{x} &= (\nabla f_i(\boldsymbol{x}^k))^T\boldsymbol{x}^k - f_i(\boldsymbol{x}^k). \end{aligned}$$

Thus for the linearized equality ,

$$\begin{aligned} \boldsymbol{A}^T &= [\nabla f_{m_I+1}(\boldsymbol{x}^k), \ldots, \nabla f_{m_I+m_E}(\boldsymbol{x}^k)] \\ \boldsymbol{b} &= \begin{bmatrix} (\nabla f_{m_I+1}(\boldsymbol{x}^k))^T\boldsymbol{x}^k - f_{m_I+1}(\boldsymbol{x}^k) \\ \vdots \\ (\nabla f_{m_I+m_E}(\boldsymbol{x}^k))^T\boldsymbol{x}^k - f_{m_I+m_E}(\boldsymbol{x}^k) \end{bmatrix} \end{aligned}$$

and we have a system of m_E linear equations given by $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

To linearize the equalities at a given iteration, the point we use, $\boldsymbol{x}^k$, is assumed to be on the surface given by the nonlinear equalities. When this point $\boldsymbol{x}^k$ is infeasible for the nonlinear equalities, we still use it for the first order Taylor series approximation but this matter needs more consideration in future. At this juncture, using even an infeasible point for linearization of the equalities does not seem to affect the results we get.

2.5 The New Algorithm

The following outline gives the key steps for using the ellipsoid algorithm for equality constrained problems. I have left out some details, about making the algorithm more numerically stable and about recentering. These aspects along with the key steps from the following algorithm are discussed in detail in §3.

We are given the following NLP with starting bounds $\boldsymbol{U}$ and $\boldsymbol{L}$.

$$\begin{aligned} \min_{\boldsymbol{x}\in\Re^n} \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \;\leq\; 0, \quad i = 1\ldots m_I \\ & f_i(\boldsymbol{x}) \;=\; 0, \quad i = m_I + 1 \ldots m_I + m_E. \end{aligned}$$

0. Find

$$\boldsymbol{x}^0 = \frac{(\boldsymbol{U}+\boldsymbol{L})}{2}$$

$$\text{and } \boldsymbol{Q}_0 = \frac{n}{4}\begin{pmatrix} (U_1-L_1)^2 & 0 & \dots & 0 \\ 0 & (U_2-L_2)^2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & (U_n-L_n)^2 \end{pmatrix}$$

1. Project $\boldsymbol{x}$ onto F, the flat of equalities.

1a. Find $\boldsymbol{A}$ and $\boldsymbol{b}$ by linearizing the equalities as in §2.4

1b. Solve $\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k$ for $\boldsymbol{\alpha}$

1c. Substitute that value of $\boldsymbol{\alpha}$ in $\boldsymbol{x}_F = \boldsymbol{x} + \boldsymbol{A}^T\boldsymbol{\alpha}$ to find $\boldsymbol{x}_F$

1d. Set $\boldsymbol{x}^k = \boldsymbol{x}_F^k$

2. Look for a violated inequality constraint, which means

$$\begin{aligned} \text{if } f_I(\boldsymbol{x}^k) &> 0, \text{ set } i = I \\ \text{if } f_i(\boldsymbol{x}^k) &\leq 0 \text{ for all } i = 1 \dots m_I, \text{ set } i = 0 \end{aligned}$$

3. Find the direction $\boldsymbol{d}$

$$\begin{aligned} \boldsymbol{g} &= \frac{\nabla f_i(\boldsymbol{x}^k)}{\|(\nabla f_i(\boldsymbol{x}^k))\|} \\ \boldsymbol{d} &= -\frac{(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}} \end{aligned}$$

4. Use $\boldsymbol{d}$ to update $\boldsymbol{x}$ and $\boldsymbol{Q}$

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \boldsymbol{x}^k + \frac{1}{n+1}\boldsymbol{d} \\ \boldsymbol{Q}_{k+1} &= \frac{n^2}{n^2-1}(\boldsymbol{Q}_k - \frac{2}{n+1}\boldsymbol{d}\boldsymbol{d}^T) \end{aligned}$$

5. Increase k by 1 and go to 1.

2.6 Geometrical Interpretation for d

It is instructive to investigate how the direction $\boldsymbol{d}$ and the cutting hyperplane H_k correspond to the ones in the original ellipsoid algorithm. We do not use the original supporting hyperplane as the cutting hyperplane in the new algorithm. This is because the tangent hyperplane parallel to the cutting hyperplane does not guarantee that the resulting direction will lie in the flat of equalities. The new direction we have found using the KKT conditions including the equality constraints does guarantee that the point $\boldsymbol{x}^k + \boldsymbol{d}$ lies in the boundary of the ellipsoid and is also in the flat of equalities.

Let us first look at the tangent hyperplane to the ellipsoid E_k (with $\boldsymbol{Q}$ as the defining matrix) at the point $\boldsymbol{x}^k + \boldsymbol{d}$ on the ellipsoid, where

$$\boldsymbol{d} = -\frac{(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}}.$$

As mentioned in §2.2, the denominator in the above formula is not zero. We know that the equation of the ellipsoid is $E_k = \{\boldsymbol{x} \in \Re^n | (\boldsymbol{x} - \boldsymbol{x}^k)^T \boldsymbol{Q}^{-1}(\boldsymbol{x} - \boldsymbol{x}^k) \leq 1\}$. The equation of a tangent hyperplane to this ellipsoid at a point $\boldsymbol{x}^k + \boldsymbol{d}$ in its boundary is [1] [7]

$$\begin{aligned}
\left(\boldsymbol{Q}^{-1}(\boldsymbol{x}^k + \boldsymbol{d} - \boldsymbol{x}^k)\right)^T (\boldsymbol{x} - \boldsymbol{x}^k - \boldsymbol{d}) &= 0 \\
(\boldsymbol{Q}^{-1}\boldsymbol{d})^T(\boldsymbol{x} - \boldsymbol{x}^k - \boldsymbol{d}) &= 0.
\end{aligned}$$

Substituting for $\boldsymbol{d}$ in $\boldsymbol{Q}^{-1}\boldsymbol{d}$ and leaving the other $\boldsymbol{d}$ in the expression alone for the sake of notational convenience,

$$\left(\boldsymbol{Q}^{-1}\left(\frac{(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}{-\sqrt{\boldsymbol{g}^T(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}}\right)\right)^T (\boldsymbol{x} - \boldsymbol{x}^k - \boldsymbol{d}) = 0.$$

Since the denominator is nonzero, we can multiply through by it, obtaining

$$
\begin{aligned}
\left(Q^{-1}\left((Q - QA^T(AQA^T)^{-1}AQ)g\right)\right)^T (x - x^k - d) &= 0 \\
\left((I - A^T(AQA^T)^{-1}AQ)g\right)^T (x - x^k - d) &= 0. \qquad (2.8)
\end{aligned}
$$

Simplifying above equation, we get

$$
\left((I - A^T(AQA^T)^{-1}AQ)g\right)^T (x - x^k) = \left((I - A^T(AQA^T)^{-1}AQ)g\right)^T d.
$$

After substituting for d the right hand side becomes

$$
\begin{aligned}
&= g^T(I - A^T(AQA^T)^{-1}AQ)^T \left(-\frac{(Q - QA^T(AQA^T)^{-1}AQ)g}{\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}}\right). \\
&= -\frac{g^T(I - QA^T(AQA^T)^{-1}A)(Q - QA^T(AQA^T)^{-1}AQ)g}{\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}}
\end{aligned}
$$

We can simplify the above numerator using (2.6)-(2.7) from §2.2 so the right hand side becomes

$$
\begin{aligned}
&= -\frac{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}{\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}} \\
&= -\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}.
\end{aligned}
$$

Thus we can write (2.8) as

$$
\left((I - A^T(AQA^T)^{-1}AQ)g\right)^T (x - x^k) = -\sqrt{g^T(Q - QA^T(AQA^T)^{-1}AQ)g}.
$$

The above equation gives the tangent hyperplane to the ellipsoid E_k at the point $x^k + d$ and should be compared to the formula (1.6) we found in the classical algorithm of §1.2.

When we use the updates in the ellipsoid algorithm for Q and x, it is assumed that the half of the ellipsoid we are trying to enclose lies between this tangent hyperplane and a parallel cutting hyperplane through the center of the ellipsoid. Until now we have not needed explicitly to find the formula for the cutting hyperplane.

For the updates we never have to use the formula for the cutting hyperplane, but it is important for understanding the geometry of this modified ellipsoid method and also for giving a plausibility argument for convergence of the new algorithm. The cutting hyperplane through the center for this algorithm is given by putting $\boldsymbol{d} = \boldsymbol{0}$ in (2.8), or

$$H_k = \{\boldsymbol{x} \in \Re^n | ((\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g})^T(\boldsymbol{x} - \boldsymbol{x}^k) = 0\}.$$

This is to be compared with the similar formula (1.5) in the classical algorithm of §1.2. Both the cutting hyperplane and the tangent hyperplane of the new algorithm can be obtained from those of the classical algorithm by simply substituting in the original formulas $\boldsymbol{Mg} = (\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}$ for $\boldsymbol{g}$. If we use the formula for $\boldsymbol{d}$ from the original ellipsoid algorithm on this modified $\boldsymbol{Mg}$, we should get the new direction $\boldsymbol{d}$ in the modified ellipsoid algorithm. In the original ellipsoid algorithm,

$$\boldsymbol{d} = -\frac{\boldsymbol{Qg}}{\sqrt{\boldsymbol{g}^T\boldsymbol{Qg}}}.$$

Substituting $\boldsymbol{Mg}$ for $\boldsymbol{g}$ in this formula yields,

$$\begin{aligned}
\boldsymbol{d} &= -\frac{\boldsymbol{Q}(\boldsymbol{Mg})}{\sqrt{(\boldsymbol{Mg})^T\boldsymbol{QMg}}} \\
\boldsymbol{d} &= -\frac{\boldsymbol{Q}(\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}{\sqrt{((\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g})^T\boldsymbol{Q}(\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}} \\
\boldsymbol{d} &= -\frac{(\boldsymbol{Q} - \boldsymbol{QA}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T(\boldsymbol{I} - \boldsymbol{QA}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{A})(\boldsymbol{Q} - \boldsymbol{QA}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}}.
\end{aligned} \tag{2.9}$$

We can simplify the denominator in (2.9) using (2.6)-(2.7) from §2.2. Thus we can say,

$$\boldsymbol{d} = -\frac{(\boldsymbol{Q} - \boldsymbol{QA}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T(\boldsymbol{Q} - \boldsymbol{QA}^T(\boldsymbol{AQA}^T)^{-1}\boldsymbol{AQ})\boldsymbol{g}}}$$

This is the same $\boldsymbol{d}$ we got using the KKT conditions.

2.7 The Volumes of Ellipsoids of Intersection

We know [9] that at any iteration the ratio of volumes of successive ellipsoids is given by

$$\frac{V(E_{k+1})}{V(E_k)} = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2} < 1,$$

so the volumes $V(E_k)$ decrease monotonically to zero. For arguing that this algorithm converges we need to prove that the volumes of the intersections of the ellipsoids with the flat F also decrease monotonically to zero.

It is clear that the intersection of the ellipsoid with a hyperplane through its center is also an ellipsoid. Let us find the ellipsoid $\tilde{E}_k = E_k \cap F$ that is the intersection of the ellipsoid $E_k = \{x \in \Re^n | (\boldsymbol{x} - \boldsymbol{x}^k)^T \boldsymbol{Q}_k^{-1}(\boldsymbol{x} - \boldsymbol{x}^k) \leq 1\}$ with the flat $F = \{\boldsymbol{x} \in \Re^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$. Let the columns of $\boldsymbol{B}$ denote a basis for the null space of $\boldsymbol{A}$. Then any solution $\boldsymbol{x}$ to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ satisfies $\boldsymbol{x} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b} + \boldsymbol{B}\boldsymbol{z}$ for some $\boldsymbol{z} \in \Re^{n-m_E}$ [17]. There is a 1-to-1 correspondence between vectors $\boldsymbol{x}$ and $\boldsymbol{z}$. The ellipsoid $\tilde{E}_k$ can be obtained by substituting for $\boldsymbol{x}$ and $\boldsymbol{x}^k$ in the above formula for the ellipsoid E_k.

$$\begin{aligned}
((\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b} + \boldsymbol{B}\boldsymbol{z}) - (\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b} + \boldsymbol{B}\boldsymbol{z}^k))^T\boldsymbol{Q}_k^{-1} & \\
((\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b} + \boldsymbol{B}\boldsymbol{z}) - (\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{b} + \boldsymbol{B}\boldsymbol{z}^k)) &\leq 1 \\
(\boldsymbol{B}\boldsymbol{z} - \boldsymbol{B}\boldsymbol{z}^k)^T\boldsymbol{Q}_k^{-1}(\boldsymbol{B}\boldsymbol{z} - \boldsymbol{B}\boldsymbol{z}^k) &\leq 1. \\
\text{Thus, } \tilde{E}_k = \{\boldsymbol{z} \in \Re^{n-m_E} | (\boldsymbol{z} - \boldsymbol{z}^k)^T\boldsymbol{B}^T\boldsymbol{Q}_k^{-1}\boldsymbol{B}(\boldsymbol{z} - \boldsymbol{z}^k) &\leq 1\}.
\end{aligned}$$

Next recall that the volume of an ellipsoid $E = \{x \in \Re^n | (\boldsymbol{x} - \boldsymbol{c})^T\boldsymbol{W}^T\boldsymbol{W}(\boldsymbol{x} - \boldsymbol{c}) \leq 1\}$ is $Vol(E) = det(\boldsymbol{W}^{-1}) \times Vol(S(0,1))$ where $S(0,1)$ is the unit ball in n dimensions (centered at the origin) [6]. An ellipsoid is of course the image of the unit ball $S(0,1)$ under some affine transformation. Armed with all the above information we can find the ratio of volumes of successive ellipsoids $\tilde{E}_k$ in the lower dimension and prove that these volumes go to zero monotonically as the number of iterations k increases.

Without loss of generality assume $E_k = S(0,1) = \{\boldsymbol{x} \in \Re^n | \boldsymbol{x}^T\boldsymbol{Q}_k^{-1}\boldsymbol{x} \leq 1\}$ where $\boldsymbol{Q}_k^{-1} = \boldsymbol{Q}_k = \boldsymbol{I}$. Here $Vol(E_k) = Vol(S(0,1))$. Having assumed that E_k

is centered at the origin, it must be that the flat F of equalities goes through the origin and $\boldsymbol{Ax} = \boldsymbol{0}$. We can further assume without loss that $\boldsymbol{Ax} = \boldsymbol{0}$ represents one of the co-ordinate hyperplanes. If $m_E = n-1$, $\boldsymbol{A}$ is the $(n-1)\times n$ matrix

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & & \ddots & 0 \\ 0 & 0 & & \dots & 1 \end{pmatrix}$$

which has maximum possible rank $n-1$. Then a basis for the null space of $\boldsymbol{A}$ is $\boldsymbol{B} = [1, 0, \dots 0]^T$. The ellipsoid of intersection, in $n-(n-1) = 1$ dimension, is $\tilde{E}_k = \{\boldsymbol{z} \in \Re^1 | \boldsymbol{z}^T \boldsymbol{B}^T \boldsymbol{I} \boldsymbol{B} \boldsymbol{z} \leq 1\}$ and $\boldsymbol{B}^T \boldsymbol{I} \boldsymbol{B} = 1$ is a scalar. If $\tilde{S}(0,1)$ is the unit ball in 1 dimension, then $Vol(\tilde{E}_k) = det(1) \times Vol(\tilde{S}(0,1)) = Vol(\tilde{S}(0,1))$.

Now let us make the only possible update for the ellipsoid in n dimensions using $\boldsymbol{d} = [1, 0, \dots 0]^T$. Using this $\boldsymbol{d}$ in the update formulas for the ellipsoid algorithm of §2.5.

$$\begin{aligned}
\boldsymbol{x}^{k+1} &= \left[\frac{1}{n+1}, 0, \dots, 0\right]^T \\
\boldsymbol{Q}_{k+1} &= \frac{n^2}{n^2-1}\left[\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ 0 & \ddots & & 0 \\ 0 & & \dots & 1 \end{pmatrix} - \frac{2}{n+1}\begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ 0 & \ddots & & 0 \\ 0 & & \dots & 0 \end{pmatrix}\right]. \\
\boldsymbol{Q}_{k+1} &= \begin{pmatrix} \frac{n^2}{(n+1)^2} & 0 & \dots & 0 \\ 0 & \frac{n^2}{n^2-1} & \dots & 0 \\ 0 & & \ddots & 0 \\ 0 & 0 & \dots & \frac{n^2}{n^2-1} \end{pmatrix} \\
\boldsymbol{Q}_{k+1}^{-1} &= \begin{pmatrix} \frac{(n+1)^2}{n^2} & 0 & \dots & 0 \\ 0 & \frac{n^2-1}{n^2} & \dots & 0 \\ 0 & & \ddots & 0 \\ 0 & 0 & \dots & \frac{n^2-1}{n^2} \end{pmatrix}
\end{aligned}$$

Now we can find the new ellipsoid in n dimensions and its volume.

$$\begin{aligned}
E_{k+1} &= \{\boldsymbol{x} \in \Re^n | (\boldsymbol{x} - \boldsymbol{x}^1)^T \boldsymbol{W}^T \boldsymbol{W} (\boldsymbol{x} - \boldsymbol{x}^1) \leq 1\} \quad \text{where} \quad \boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{Q}_{k+1}^{-1}. \\
Vol(E_{k+1}) &= det(\boldsymbol{W}^{-1}) \times Vol(S(0,1)) \\
&= \sqrt{det(\boldsymbol{Q}_{k+1})} \times Vol(S(0,1)) \\
&= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2} \times Vol(S(0,1)).
\end{aligned}$$

For the lower-dimensional ellipsoid, $\tilde{E}_{k+1} = \{\boldsymbol{z} \in \Re^1 | \boldsymbol{z}^T \boldsymbol{B}^T \boldsymbol{Q}_{k+1}^{-1} \boldsymbol{B}\boldsymbol{z} \leq 1\}$ where

$$\begin{aligned}
\boldsymbol{B}^T\boldsymbol{Q}_{k+1}^{-1}\boldsymbol{B} &= [1,0,\ldots,0]\begin{pmatrix} \frac{(n+1)^2}{n^2} & 0 & \ldots & 0 \\ 0 & \frac{n^2-1}{n^2} & \ldots & 0 \\ 0 & & \ddots & 0 \\ 0 & 0 & \ldots & \frac{n^2-1}{n^2} \end{pmatrix}\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\
&= \frac{(n+1)^2}{n^2}, \\
&= \left(\frac{n+1}{n}\right)\left(\frac{n+1}{n}\right) \\
\text{and} \quad Vol(\tilde{E}_{k+1}) &= \frac{n}{n+1} \times Vol(\tilde{S}(0,1)).
\end{aligned}$$

Thus the ratio of the volumes of the ellipsoids of intersection at each iteration when rank$(\boldsymbol{A}) = n-1$ is

$$\begin{aligned}
\frac{Vol(\tilde{E}_{k+1})}{Vol(\tilde{E}_k)} &= \frac{\frac{n}{n+1} \times Vol(\tilde{S}(0,1))}{Vol(\tilde{S}(0,1))} \\
&= \frac{n}{n+1} < 1.
\end{aligned}$$

After k iterations, volume will have been reduced by the factor $\left(\frac{n}{n+1}\right)^k$. As k increases this ratio goes to 0.

Next let us consider the case where rank$(\boldsymbol{A}) = n-2$. As in the previous case assume $F = \{\boldsymbol{x} \in \Re^n | \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}\}$ so that $\boldsymbol{A}$ is the $(n-2) \times n$ matrix

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & & \ddots & 0 \\ 0 & 0 & 0 & & \dots & 1 \end{pmatrix}.$$

Now a basis for the null space of $\boldsymbol{A}$ is given by

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \end{bmatrix}^T.$$

The ellipsoid of intersection in $n - (n-2) = 2$ dimension is

$$\begin{aligned} \tilde{E}_k &= \{\boldsymbol{z} \in \Re^2 | \boldsymbol{z}^T \boldsymbol{B}^T \boldsymbol{I} \boldsymbol{B} \boldsymbol{z} \le 1\} \text{ with } \boldsymbol{B}^T\boldsymbol{I}\boldsymbol{B} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ \text{so } Vol(\tilde{E}_k) &= det \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \times Vol(\tilde{S}(0,1)) \\ Vol(\tilde{E}_k) &= Vol(\tilde{S}(0,1)). \end{aligned}$$

Here $\tilde{S}(0,1)$ is the unit ball in 2 dimensions. Without loss of generality again let $\boldsymbol{d} = [1, 0, \dots 0]^T$. Then $\tilde{E}_{k+1} = \{\boldsymbol{z} \in \Re^2 | \boldsymbol{z}^T \boldsymbol{B}^T \boldsymbol{Q}_{k+1}^{-1} \boldsymbol{B} \boldsymbol{z} \le 1\}$ and $\boldsymbol{Q}_{k+1}^{-1}$ is the same as in the previous case. Here

$$\begin{aligned} \boldsymbol{B}^T \boldsymbol{Q}_{k+1}^{-1} \boldsymbol{B} &= \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \end{bmatrix} \begin{pmatrix} \frac{(n+1)^2}{n^2} & 0 & \dots & 0 \\ 0 & \frac{n^2-1}{n^2} & \dots & 0 \\ 0 & & \ddots & 0 \\ 0 & 0 & \dots & \frac{n^2-1}{n^2} \end{pmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix} \\ \boldsymbol{B}^T \boldsymbol{Q}_{k+1}^{-1} \boldsymbol{B} &= \begin{pmatrix} \frac{(n+1)^2}{n^2} & 0 \\ 0 & \frac{n^2-1}{n^2} \end{pmatrix} \text{ and} \\ Vol(\tilde{E}_{k+1}) &= \frac{n}{n+1} \left(\frac{n^2}{n^2-1} \right)^{1/2} \times Vol(\tilde{S}(0,1)). \end{aligned}$$

Thus the ratio of volumes of successive ellipsoids of intersection at each iteration when rank($\boldsymbol{A}$) $= n-2$ is

$$\begin{aligned} \frac{Vol(\tilde{E}_{k+1})}{Vol(\tilde{E}_k)} &= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{1/2} < 1 \\ \text{and} \quad \frac{Vol(\tilde{E}_{k+1})}{Vol(\tilde{E}_0)} &= \left(\frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{1/2}\right)^k. \end{aligned}$$

As k increases the above ratio goes to 0. For the other two possible directions $\boldsymbol{d}$ for this case, namely $\boldsymbol{d} = [0, 1, 0, \ldots, 0]^T$ and $\boldsymbol{d} = [1/\sqrt{2}, 1/\sqrt{2}, 0, \ldots, 0]^T$, the volume of the new ellipsoid $\tilde{E}_{k+1}$ is the same as we found above.

In general, for rank($\boldsymbol{A}$) $= n - m$, where $1 \leq m \leq (n-1)$, the ratio of the voiumes of successive ellipsoids of intersection is

$$\begin{aligned} \frac{Vol(\tilde{E}_{k+1})}{Vol(\tilde{E}_k)} &= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-(n-m)-1)/2} \\ &= \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(m-1)/2} \end{aligned}$$

For problems where rank($\boldsymbol{A}$) $= n-1$, $m = 1$ and the above ratio is $\dfrac{n}{n+1}$ which is strictly less than 1. For problems where rank($\boldsymbol{A}$) $= 1$, $m = n-1$ and the above ratio takes on its largest value,

$$\frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-2)/2}$$

As mentioned earlier this is also less than 1. Hence the ratio of successive volumes is strictly less than 1 for all $n \geq 2$ (we only consider problems with equality constraints for $n \geq 2$). This proves that the volumes of the ellipsoids of intersection decrease monotonically to zero as k increases.

2.8 A Plausibility Argument for Convergence

In the argument below, we assume that the objective function and all inequality constraints are convex and that the equalities are linear. The object of the algorithm is to locate an optimal point x^* that solves NLP, which is given by

$$\begin{aligned} \min_{x \in \Re^n} \quad & f_0(x) \\ \text{subject to} \quad f_i(x) & \leq 0, \quad i = 1 \ldots m_I \\ f_i(x) & = 0, \quad i = m_I + 1 \ldots m_I + m_E. \end{aligned}$$

Therefore, for any particular instance of NLP, the algorithm is said to converge to x^* relative to a given accuracy limit $\tau > 0$ if there exists a k such that $\|x^* - x^k\| \leq \tau$.

A discussion of the volume of the feasible set, S, is important at this point. We have assumed that S' is a full-dimensional feasible set. We only use S' for making feasibility cuts on the ellipsoid, taking into consideration only violated inequality constraints. We make sure that each new center x^k is feasible for the equality constraints by modifying the direction d and projecting the center onto the flat F as discussed in §2.3.

In the discussion that follows, whenever I mention any x as feasible (or infeasible) I am talking in terms of the inequality constraints alone. Feasibility for the equality constraints is guaranteed by the direction d in §2.5. The following hypotheses together give us the plausibility argument for the convergence of this modified ellipsoid algorithm.

(1) The optimal point x^* is contained in each of the ellipsoids E_k generated by the algorithm in §2.5.

(2) $V(E_{k+1})/V(E_k) = c_n < 1$

(3) (1)and (2) imply convergence in the sense described above.

Let us consider these points in reverse order. Suppose (1) and (2) are true and assume that (3) does not hold; that is, $x^k \to x^\infty \neq x^*$. The two cases where this could be true while the volumes of the E_k generated by the modified ellipsoid algorithm decrease in geometric progression are, for the volumes of intersection of the ellipsoids E_k with the flat F to not decrease monotonically, or for the E_k to approach an ellipsoid of dimension less than n, having $\boldsymbol{x}^{\infty}$ at its center and containing $\boldsymbol{x}^*$. The following arguments show why neither of these two cases can happen for the algorithm we are using.

Having a full-dimensional feasible set is a necessary condition for the ellipsoid algorithm to work. The ellipsoids we generate in the new algorithm (in §2.5) consider only the full dimensional part S' of the feasible set S. Using the modified $\boldsymbol{d}$ we make sure that the centers of the ellipsoids are on the flat F. Thus all the feasibility cuts are generated only for S' and the E_k are ellipsoids of dimension n. In §2.7 I proved that the volumes of the ellipsoids of intersection $\tilde{E}_k$ decrease monotonically to zero. Since f_0 is convex, then $\boldsymbol{g}_0(\boldsymbol{x}^{\infty})^T(\boldsymbol{x}^* - \boldsymbol{x}^{\infty}) < 0$ and if an optimization cut were made, $\boldsymbol{x}^k$ would move towards $\boldsymbol{x}^*$. Thus it must be that for all k greater than some k^+, no optimization cuts are made. Since an optimization cut is made whenever $\boldsymbol{x}^k$ is feasible, this implies that for $k > k^+$, no $\boldsymbol{x}^k$ is feasible. This means that for all $k > k^+$, $S' \bigcap E_k = \emptyset$. This is impossible because (1) states that the optimal point $\boldsymbol{x}^*$ is contained in each of the ellipsoids E_k and we know that $\boldsymbol{x}^* \in S$. Thus (3) is established as a consequence of (1) and (2) for the problem in which S' has nonzero volume relative to R^n.

Now consider proposition (2). Given the starting ellipsoid E_0, the subsequent ellipsoids are the Löwner-John ellipsoids of the appropriate half of the current ellipsoid. As mentioned in §1.3, the ratio of their volumes does not depend on k and is given by

$$\frac{V(E_{k+1})}{V(E_k)} = \frac{n}{n+1}\left(\frac{n^2}{n^2-1}\right)^{(n-1)/2} < 1.$$

It remains to show only (1), namely that each E_k contains $\boldsymbol{x}^*$. The algorithm initialization step supplies an E_0 containing $\boldsymbol{x}^*$. Assume for induction that $\boldsymbol{x}^* \in E_k$. By the geometry of the construction of E_{k+1},

$$E_{k+1} \supset E_k \bigcap \{\boldsymbol{x} \in \Re^n | (\boldsymbol{Mg})^T(\boldsymbol{x} - \boldsymbol{x}^k) \leq 0\}.$$

Using the hyperplane H_k as described in §2.6, we cut the ellipsoid E_k in two halves and we need to show that the half containing the optimal point is the one we enclose by the new ellipsoid E_{k+1}. Thus it is only necessary to show that $(\boldsymbol{M}\boldsymbol{g})^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \le 0$.

We can simplify the above expression by using $\boldsymbol{M} = \boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q}$ from §2.6 in the following way.

$$\begin{aligned}
(\boldsymbol{M}\boldsymbol{g})^T(\boldsymbol{x}^* - \boldsymbol{x}^k) &= ((\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g})^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
&= \boldsymbol{g}^T(\boldsymbol{I} - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
&= \boldsymbol{g}^T(\boldsymbol{I} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A})(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
&= (\boldsymbol{g}^T - \boldsymbol{g}^T\boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A})(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
&= \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) - \boldsymbol{g}^T\boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
&= \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) - \boldsymbol{g}^T\boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}(\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{A}\boldsymbol{x}^k)
\end{aligned}$$

For $\boldsymbol{x}^*$ to be the optimal point, it needs to be feasible for both the equality and inequality constraints, which means $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$. Also at each iteration k, the center of the ellipsoid E_k lies in the flat of equalities F, giving us $\boldsymbol{A}\boldsymbol{x}^k = \boldsymbol{b}$. Thus the above expression becomes

$$\begin{aligned}
(\boldsymbol{M}\boldsymbol{g})^T(\boldsymbol{x}^* - \boldsymbol{x}^k) &= \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) - \boldsymbol{g}^T\boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}(\boldsymbol{b} - \boldsymbol{b}) \\
&= \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) - \boldsymbol{0} \\
&= \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k).
\end{aligned}$$

Using this simplified expression, we only need to prove $\boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \le 0$. Now we need to consider the two methods we use to find $\boldsymbol{g}$ as appropriate, namely the violated constraint cut and the objective function cut.

Case 1. A constraint function cut is made at $\boldsymbol{x}^k \notin S'$ using the normalized gradient $\boldsymbol{g} = \nabla f_i(\boldsymbol{x}^k)/\|\nabla f_i(\boldsymbol{x}^k)\|$. Here $\nabla f_i(\boldsymbol{x}^k) \ne 0$ since f_i is convex, $\boldsymbol{x}^k \notin S'$, and

$S' \neq \emptyset$. Because f_i is a convex differentiable function,

$$f_i(\boldsymbol{x}) \geq f_i(\boldsymbol{x}^k) + \nabla f_i(\boldsymbol{x}^k)^T(\boldsymbol{x} - \boldsymbol{x}^k).$$

In particular, for $\boldsymbol{x} = \boldsymbol{x}^*$

$$\begin{aligned}
f_i(\boldsymbol{x}^*) &\geq f_i(\boldsymbol{x}^k) + \nabla f_i(\boldsymbol{x}^k)^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
f_i(\boldsymbol{x}^*) - f_i(\boldsymbol{x}^k) &\geq \nabla f_i(\boldsymbol{x}^k)^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \\
\frac{f_i(\boldsymbol{x}^*) - f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|} &\geq \left(\frac{\nabla f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|}\right)^T (\boldsymbol{x}^* - \boldsymbol{x}^k) \\
\text{and thus } \boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) &\leq \frac{f_i(\boldsymbol{x}^*) - f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|}.
\end{aligned}$$

Because $\boldsymbol{x}^k \notin S'$, $f_i(\boldsymbol{x}^*) < f_i(\boldsymbol{x}^k)$ and $\boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \leq 0$ as required.

Case 2. An objective function cut is made at $\boldsymbol{x}^k \in S'$ using the normalized gradient $\boldsymbol{g} = \nabla f_0(\boldsymbol{x}^k)/\|\nabla f_0(\boldsymbol{x}^k)\|$. Recall that since f_0 is convex and $\boldsymbol{x}^k \neq \boldsymbol{x}^*$, $\nabla f_0(\boldsymbol{x}^k) \neq 0$. From convexity of f_0, as in Case 1,

$$\boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \leq \frac{f_0(\boldsymbol{x}^*) - f_0(\boldsymbol{x}^k)}{\|\nabla f_0(\boldsymbol{x}^k)\|}.$$

Because $\boldsymbol{x}^k \neq \boldsymbol{x}^*$ and $\boldsymbol{x}^k \in S'$, $f_0(\boldsymbol{x}^*) \leq f_0(\boldsymbol{x}^k)$ and $\boldsymbol{g}^T(\boldsymbol{x}^* - \boldsymbol{x}^k) \leq 0$ as required.

CHAPTER 3
Implementation

This chapter presents and discusses the classical FORTRAN code that was written to implement the algorithm of §2.5.

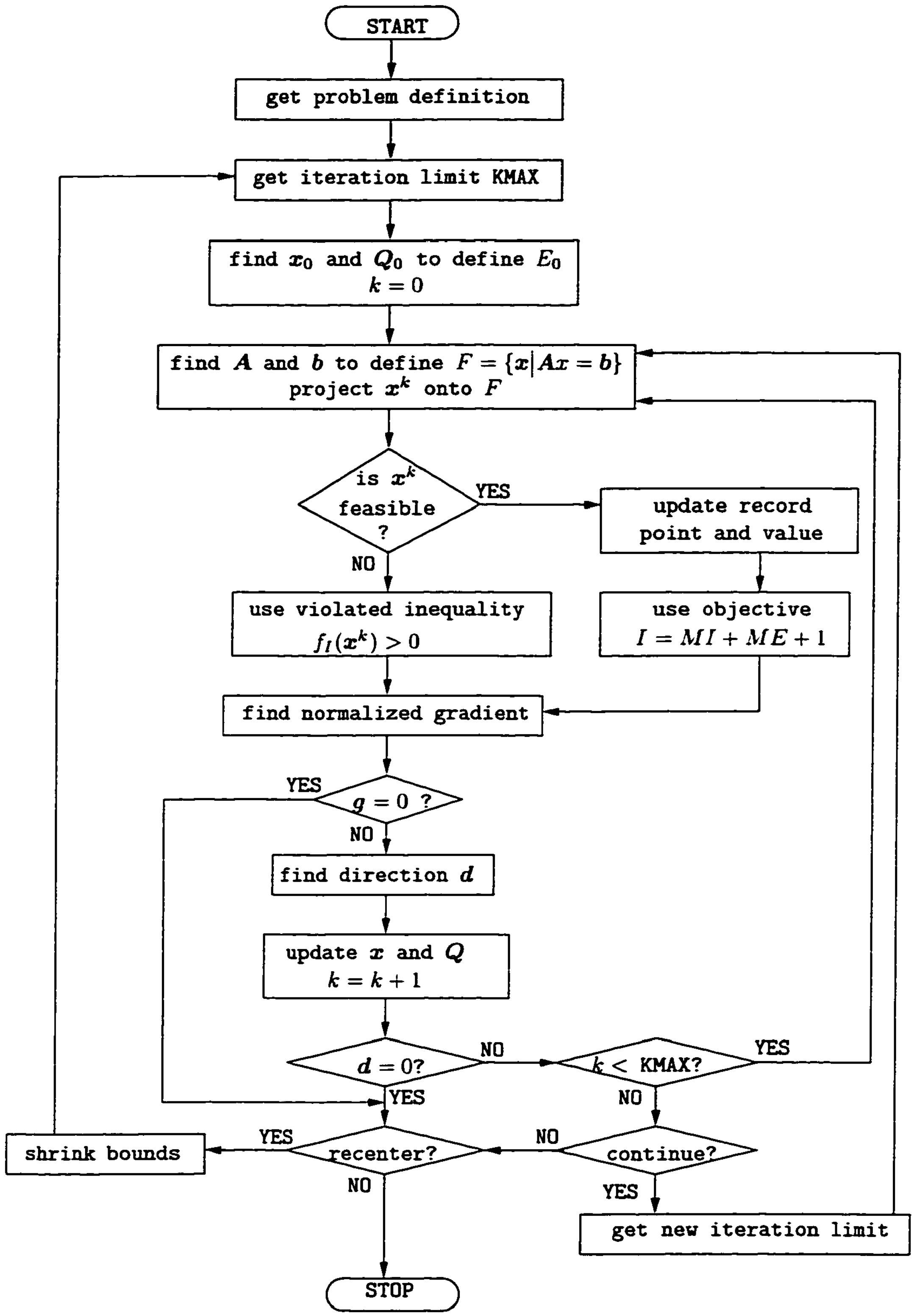
3.1 Outline of the Program

A practical implementation of the algorithm must include several refinements that are not suggested by the theory discussed earlier but which turn out to be important for user convenience or for controlling roundoff errors. No commercial routines are used anywhere in the implementation.

For problems with equality constraints, we first linearize the equalities at every iteration to find a system of linear equalities given by $\boldsymbol{Ax} = \boldsymbol{b}$. Depending on the constraints, this Taylor series approximation does not always give the true representation of the surface given by the equalities. Thus, although we project each point onto the flat of equalities before using it, this does not guarantee that the point is feasible for equality constraints that are nonlinear. Therefore in keeping record of the best point found, before accepting a point as a record point we make sure it satisfies the actual equality constraints within a certain tolerance.

Depending on the nature of the constraints, repeated EA cuts can cause the ellipsoids to become highly aspheric or in other words very long and skinny, and thus the optimal point may still be in the ellipsoid but for numerical reasons we may not be able to reach it. To overcome this problem, we use recentering. The program recenters by using the record point as the new initial point, or if no record point has been found yet the current point is the new initial point. The new bounds are centered on this point and separated by a certain fraction of the distance between the original bounds on $\boldsymbol{x}$. In effect we restart with a better center and a smaller initial ellipsoid than before. When we recenter, we reset the iteration counter to zero.

In finding the projection and the new direction $\boldsymbol{d}$ we never use an inverse of a matrix for our calculations; instead we solve systems of linear equations. The flowchart on the following page summarizes the calculation.


START
get problem definition
get iteration limit KMAX
find x_0 and Q_0 to define E_0
$k = 0$
find A and b to define $F = \{x | Ax = b\}$
project x^k onto F
is x^k feasible ?
YES
update record point and value
NO
use violated inequality
$f_l(x^k) > 0$
use objective
$l = MI + ME + 1$
find normalized gradient
YES
$g = 0$?
NO
find direction d
update x and Q
$k = k + 1$
$d = 0$?
NO
$k <$ KMAX?
YES
YES
NO
shrink bounds
YES
recenter?
NO
continue?
NO
YES
get new iteration limit
STOP

3.2 The Driver

The calculation described by the flowchart of §3.1 was implemented by the main program that is listed in pieces on the next few pages.

```
C
Code by Sharmila  Shah
C
C     This program implements the algorithm for solving
C     equality-constrained problems using the ellipsoid
C     method.
C
C     variable  meaning
C     --------  -------
C     D         direction vector
C     DABS      Fortran function gives |REAL*8|
C     DF        projection of D onto the flat of equalities
C     DFLOAT    Fortran function gives REAL*8 for INTEGER*4
C     DIRECT    routine solves KKT to get the next D
C     DSQRT     Fortran function gives sqrt of a REAL*8
C     F         the current objective value
C     FCN       subprogram returns the value of the I'th function
C     FR        the record objective value
C     G         gradient, then normalized gradient
C     GETEXP    routine gets the experimental parameters
C     GMAX      gradient component largest in absolute value
C     GRAD      subprogram returns the gradient of the I'th function
C     I         index of violated constraint, or objective
C     II        index on the inequalities
C     J         index on the variables
C     JJ        second index on the variables
C     K         iteration counter
C     KMAX      limit on iterations
C     KNEW      next number of iterations specified by the user
C     KNOREC    T => know a record value and point
C     LVCZG     index of previous violated constraint with zero grad
C     ME        number of equalities
C     MI        number of inequalities
C     N         number of variables
C     NVCZG     number of violated constraints with zero gradient
C     OK        T => DF is not zero yet
C     OK1       T => X can be projected
C     P         constant in ellipsoid update formula(=n^2/n^2-1)
C     PRJECT    routine projects X onto the flat of equalities
C     PROMPT    routine prompts for input from the keyboard
C     Q         inverse matrix of the ellipsoid
C     QUERY     routine asks a yes-or-no question
C     R         constant in ellipsoid update formula(=1/n+1)
C     SW3       switches tell what is known in /NGC3/
C     T         tolerance for satisfying equality constraints
C     W         width between upper and lower bounds for this var
C     X         current ellipsoid center
```

```
48 C    XH        upper bounds on the variables
49 C    XL        lower bounds on the variables
50 C    XR        the best feasible point found so far
51 C
52 C    receive problem data from common
53      COMMON /NGC3/ SW3,N,MI,ME,XH,XL
54      LOGICAL*4 SW3(5)
55      REAL*8 XH(50),XL(50)
56 C
57 C    declare local variables
58      REAL*8 X(50),Q(50,50)/2500*0.D0/,FCN,G(50),GMAX,D(50)
59      REAL*8 DF(50),P,R,F,FR/1.D+100/,XR(50),W
60      REAL*8 T/1.D-6/
61 C
62      LOGICAL*4 QUERY,OK,OK1,KNOREC/.FALSE./
63 C
64 C    ----------------------------------------------------------------
```

The program begins with the above table of variable definitions $\boxed{\texttt{10-50}}$ followed by declarations for some problem data and for those local variables that need to be typed or dimensioned or initialized. All of the real variables in the program are REAL*8, and the arrays are dimensioned for problems having up to 50 variables.

The object code for this program must be linked with the object code for an FCN function subprogram and a GRAD subroutine that together define the objective and constraint functions and their gradients. In addition to FCN and GRAD, each problem definition code includes a BLOCK DATA subprogram that initializes a variety of problem data in common blocks. This main program uses the number of variables N, the number of inequality constraints MI, the number of equality constraints ME, and the upper and lower bounds XH and XL on the variables, which are stored in /NGC3/. (We used U and L respectively to denote the upper and lower bounds on x in the algorithms of §1.2 and the theory part §2.5.) The elements of SW3 are set by the GETEXP routine discussed below, to indicate which items of data in /NGC3/ are known.

```
65 C
66 C     get the starting point
67       CALL GETEXP(X)
68       P=DFLOAT(N**2)/DFLOAT(N**2-1)
69       R=1.DO/DFLOAT(N+1)
70 C
71 C     get the iteration limit
72     2 CALL PROMPT('iteration limit:',16)
73       READ(5,*,END=1) KMAX
74       IF(KMAX.LT.0) GO TO 2
75       IF(KMAX.EQ.0) GO TO 1
76       K=0
77 C
78 C     find the starting point and ellipsoid
79       DO 3 J=1,N
80          DO 4 JJ=1,N
81             IF (J.EQ.JJ) THEN
82                Q(J,JJ)=0.25DO*DFLOAT(N)*((XH(J)-XL(J))**2)
83             ELSE
84                Q(J,JJ)=0.DO
85             ENDIF
86     4    CONTINUE
87     3 CONTINUE
88       WRITE(6,901) (X(J),J=1,N)
89   901 FORMAT('starting X=',5(1X,1PD15.8))
```

GETEXP fills in the information in /NGC3/, and returns in X the midpoint of the given bounds. The user is asked to give the initial number of iterations. As specified in §2.5 we find the inverse matrix Q_0 $\boxed{\text{79-87}}$ for the initial ellipsoid from the bounds U and L.

$$Q_0 = \frac{n}{4}\begin{pmatrix} (U_1 - L_1)^2 & 0 & \dots & 0 \\ 0 & (U_2 - L_2)^2 & \dots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & (U_n - L_n)^2 \end{pmatrix}.$$

The inverse matrix Q of the enveloping ellipsoids is kept unfactored and in full matrix storage mode. The other matrices needed in the calculations are also kept in full storage mode.

```
C
C     project X onto the flat
   17 CALL PRJECT(X, OK1)
      IF(.NOT.OK1) THEN
      PRINT *, 'choose a different X'
      GOTO 20
      ENDIF
```

The subroutine PRJECT projects the current center onto the flat of equalities. This is required in case the equalities are nonlinear, because the linearization process at each point might pull the point away from the surface of equalities. In case of linear equalities, we use the projection so that roundoff errors do not cause the current point to stray off the flat of equalities. In an analytical solution for linear equalities, or using perfect arithmetic, the center would always stay on the flat of equalities and there would be no need for projection.

```
C
C     find a violated inequality, or use the objective
      I=0
      IF(MI.EQ.0) GO TO 6
      NVCZG=0
      LVCZG=0
   12 DO 7 II=1,MI
         I=II+1
         IF(I.GT.MI) I=1
         IF(FCN(X,N,I).GT.0.D0) GO TO 8
    7 CONTINUE
C     use the objective instead
    6 I=MI+ME+1
```

Now we determine whether the current center is feasible for the inequality constraints. If it is infeasible for some inequality constraint, we use the gradient of that violated inequality; if it is feasible or if there are no inequality constraints, then we use the gradient of the objective. In subprogram FCN, all of the inequalities are stated in $f_i(\boldsymbol{x}) \leq 0$ form as in NLP. At each iteration we reset the number NVCZG of violated constraints with zero gradient and the index LVCZG of the last violated constraint. This way, if a violated constraint has zero gradient, we can consider other violated inequality constraints. The index of the objective function is I=MI+ME+1 (where MI is the number of inequality constraints and ME is the number of equality constraints) corresponding to $i = 0$ in NLP.

```
110 C
111 C     keep track of the record value and point
112       DO 9 II=MI+1,MI+ME
113          IF(DABS(FCN(X,N,II)).GT.T) GO TO 8
114     9 CONTINUE
115       KNOREC=.TRUE.
116       F=FCN(X,N,I)
117       IF(F.LE.FR) THEN
118          FR=F
119          DO 10 J=1,N
120             XR(J)=X(J)
121   10     CONTINUE
122       ENDIF
```

The ellipsoid algorithm is capable of finding a good point very early on during its iterations. However, it is a space confinement method so it can also go away from that point in subsequent iterations and any current point $\boldsymbol{x}^k$ is not necessarily the best point found so far. For this reason it helps to keep a record of the best point found so far. We call this point the record point and its objective value the record value. If the current point is feasible for the inequality constraints then we check whether it satisfies the equality constraints within a tolerance T. If it does, then we check to see if this point is the best point we have found so far and if so update the record point and record value.

```
123 C
124 C     find the gradient for the cut
125     8 CALL GRAD(X,N,I, G)
126       GMAX=0.DO
127       DO 11 J=1,N
128          GMAX=GMAX+G(J)**2
129    11 CONTINUE
```

Using the subroutine GRAD we find $\boxed{125}$ the gradient of the appropriate function. In the algorithm in §2.5, the $\boldsymbol{g}$ we use is the normalized gradient. It is given by

$$\boldsymbol{g} = \frac{\nabla f_i(\boldsymbol{x}^k)}{\|\nabla f_i(\boldsymbol{x}^k)\|}.$$

Here we first calculate the norm GMAX which is called $\|\nabla f_i(\boldsymbol{x}^k)\|$ in the algorithm statement. This GMAX is now used to see if the gradient we are using is zero.

```
      IF(GMAX.EQ.0.D0) THEN
         IF(I.EQ.MI+ME+1) THEN
            PRINT *,'stopping with zero objective gradient'
            GO TO 1
         ELSE IF(I.EQ.LVCZG .OR. NVCZG.EQ.MI) THEN
            PRINT *,'stopping on a zero constraint gradient'
            GO TO 1
         ELSE
            LVCZG=I
            NVCZG=NVCZG+1
            GO TO 12
         ENDIF
      ENDIF
      GMAX=1.D0/DSQRT(GMAX)
      DO 13 J=1,N
         G(J)=G(J)*GMAX
   13 CONTINUE
C
      DO 14 J=1,N
         D(J)=-G(J)
   14 CONTINUE
```

If the function we are using is the objective function and has zero gradient then we have solved the problem and we can print the record point. If the function we are using is one of the inequality constraints and has zero gradient, and the inequality constraints list is not exhausted, we go back and search for another violated inequality. If the list is exhausted, then we print out the record point and invite the user to request recentering. If the gradient is not zero, we scale it and use its negative for finding the direction vector DF.

```
C
C     find the direction vector in the flat of equalities
      CALL DIRECT(Q,D,X, DF,OK)
C
C     stop if DF=0
      IF(.NOT.OK) THEN
         PRINT *,'stopping with DF=0'
         GO TO 1
      ENDIF
C
C     update X and Q when DF is not zero.
      DO 15 J=1,N
         X(J)=X(J)+R*DF(J)
         DO 16 JJ=1,N
            Q(J,JJ)=P*(Q(J,JJ)-2.D0*R*(DF(J)*DF(JJ)))
   16    CONTINUE
   15 CONTINUE
      K=K+1
```

Subroutine DIRECT, which is described later, finds the direction DF ($\boldsymbol{d}$ in the algorithm), which is used to update $\boldsymbol{x}$ and $\boldsymbol{Q}$. DIRECT also returns a flag OK that is true if DF is nonzero. If DF is zero, it means that either we are very close to the optimal point or the optimal point cannot be found from the current point. In either case we print the record value and invite the user to request recentering.

```
C
C     are we done?
      IF(K.LT.KMAX) GO TO 17
C
C     want to continue?
      IF(.NOT.QUERY('continue?',9)) GO TO 1
   18 WRITE(6,900) K
  900 FORMAT('new higher KMAX (>',I6,'):',$)
      READ *,KNEW
      IF(KNEW.GT.KMAX) THEN
         KMAX=KNEW
         GO TO 17
      ELSE
         PRINT *,'no, higher!'
         GO TO 18
      ENDIF
```

As long as neither the gradient nor DF is zero, we continue for the number of iterations specified by the user. After those iterations are over, the user is invited to request more iterations.

```
C
    1 WRITE(6,922) K, FR,(XR(J),J=1,N)
  922 FORMAT('K=',I4 ,',', ' FR=',1PD23.16/
     ;        'XR=',5(1X,1PD23.16))
      IF(QUERY('recenter?',9)) THEN
         DO 19 J=1,N
            W=XH(J)-XL(J)
            IF(KNOREC) THEN
               X(J)=XR(J)
               XH(J)=XR(J)+0.4D0*W
               XL(J)=XR(J)-0.4D0*W
            ELSE
               XH(J)=X(J)+0.4D0*W
               XL(J)=X(J)-0.4D0*W
            ENDIF
   19    CONTINUE
         GO TO 2
      ELSE
   20    STOP
      ENDIF
      END
```

Once we stop with a zero DF or a zero gradient, the user is asked if recentering is to be performed. If the answer to this query is yes, we use eighty per cent of the original width between the lower and upper limits on each x_j to determine new bounds centered on the record point, or on the current point if no record point has yet been found.

In the experimental code, there are no automatic termination criteria for stopping this process. The user stops the program, by replying no to the question about doing more iterations, when a sufficiently precise approximation to the minimizing point has been found.

3.3 Finding the Direction *d*

```
C
Code by Sharmila Shah
C
      SUBROUTINE DIRECT(Q,D,X, DF,OK)
C     This routine projects D onto the flat of the equalities.
C
C     variable  meaning
C     --------  -------
C     AQ        matrix A*Q
C     AQAT      matrix A*Q*AT (me*me)
C     AT        transpose of A
C     D         the direction vector
C     DF        the projected direction vector
C     DSQRT     finds square root for the double precision.
C     GRAD      subprogram gradient of the I'th function.
C     I         index on equalities
C     II        second index on equalities
C     J         index on variables
C     JJ        second index on variables
C     K         index for matrix multiplication
C     LOSOLV    routine solves a linear system of the form Lx=b
C     MATFAC    routine factors a matrix
C     ME        number of equality constraints
C     MI        number of inequalities
C     N         number of variables in the problem
C     OK        T => DF is not zero yet
C     Q         the matrix defining ellipsoid at given iteration
C     QLOW      the matrix defining ellipsoid at lower dimension
C     RC        return code from MATFAC; 0 => ok
C     S         interim scaling factor
C     SF        the scaling factor
C     SW3       switches tell what is known in /NGC3/
C     W         b-A*(x+D)
C     X         the current ellipsoid center
C     Y         the matrix Y=Q*AT*inv(HT)
C     YT        transpose of Y
C     YYT       matrix Y*YT
C
C     formal parameters
      REAL*8 Q(50,50),D(50),X(50),DF(50)
      LOGICAL*4 OK
C
C     receive problem data
      COMMON /NGC3/ SW3,N,MI,ME
      LOGICAL*4 SW3(5)
C
C     local variables
      REAL*8 AT(50,50),AQ(50,50),AQAT(50,50),YT(50,50),YYT(50,50)
      REAL*8 S,QLOW(50,50),SF
      INTEGER*4 RC
C
C ----------------------------------------------------------------
```

We use the subroutines `MATFAC` and `LOSOLV` in this subroutine. `MATFAC` factors a positive definite symmetric matrix into lower and upper triangular matrices and overwrites the original matrix with the factors. The lower triangular matrix from `MATFAC` is then used by `LOSOLV` to solve the system of n linear equations that we need. Both these subroutines are used in order to avoid calculating the inverses of the matrices required in the calculation of $\boldsymbol{d}$ or `DF`.

```
258 C
259 C     assume DF will be nonzero
260       OK=.TRUE.
261 C
262 C     if there are no equalities, use the original Ellipsoid Alg.
263 C     for that we make YYT a zero matrix so QLOW=Q.
264       IF(ME.EQ.0) THEN
265          DO 1 J=1,N
266             DO 2 JJ=1,N
267                YYT(J,JJ)=0
268     2       CONTINUE
269     1    CONTINUE
270          GO TO 3
271       ENDIF
```

If the problem has no equality constraints then we use the $\boldsymbol{d}$ from the original ellipsoid algorithm. For that we will use the `Q` in the original ellipsoid algorithm instead of the `QLOW` in this algorithm.

```
272 C
273 C     evaluate A'
274       DO 4 I=1,ME
275          CALL GRAD(X,N,MI+I, AT(1,I))
276     4 CONTINUE
```

We want to find $\boldsymbol{A}$ that defines the flat F at a given point. A first-order Taylor series expansion is used to find $\boldsymbol{A}$. The rows of $\boldsymbol{A}$ are the gradients $\nabla f_i(\boldsymbol{x})$, $i = m_i + 1 \cdots m_I + m_E$, which are returned here by $\boxed{275}$ `GRAD` into the columns of `AT`, the transpose of `A`.

```
277 C
278 C     find AQ
279       DO 5 J=1,ME
280          DO 6 JJ=1,N
281             AQ(J,JJ)=0.DO
282             DO 7 K=1,N
283                AQ(J,JJ)=AQ(J,JJ)+AT(K,J)*Q(K,JJ)
284     7       CONTINUE
285     6    CONTINUE
286     5 CONTINUE
287 C
288 C     find AQAT
289       DO 8  I=1,ME
290          DO 9  II=1,ME
291             AQAT(I,II)=0.DO
292             DO 10  K=1,N
293                AQAT(I,II)=AQAT(I,II)+AQ(I,K)*AT(K,II)
294    10       CONTINUE
295     9    CONTINUE
296     8 CONTINUE
```

Using AT, we calculate AQAT, or AQA^T from the algorithm. Notice that in the loop $\boxed{\text{282-284}}$ AT(K,J) is the element (J,K) of $\boldsymbol{A}$. As mentioned before, we do not find the inverse of this matrix as needed in the algorithm, but we use the subprograms MATFAC and LOSOLV and solve systems of linear equations instead and get the desired result as explained in the following piece of code and accompanying text.

```
297 C
298 C     Factorize AQAT into AQAT=H*HT then use HT
299 C     to find Y by solving n linear systems H(YT)_J=(AQ)_J
300       IF(ME.GT.1) THEN
301          CALL MATFAC(AQAT,50,ME,RC)
302          IF(RC.NE.0) THEN
303             DO 11 J=1,N
304                DF(J)=0.D0
305    11       CONTINUE
306             OK=.FALSE.
307             RETURN
308          ENDIF
309          DO 12  J=1,N
310             CALL LOSOLV(AQAT,50,ME,AQ(1,J),YT(1,J))
311    12    CONTINUE
312 C
313 C     Find YYT
314          DO 13  J=1,N
315             DO 14 JJ=1,N
316                YYT(J,JJ)=0.D0
317                DO 15 K=1,ME
318                   YYT(J,JJ)=YYT(J,JJ)+YT(K,J)*YT(K,JJ)
319    15          CONTINUE
320    14       CONTINUE
321    13    CONTINUE
322       ELSE
323 C     one equality constraint
324 C     If the scalar AQAT is zero in case of nonlinear equalities,
325 C     DF = 0; otherwise we can find YYT and QLOW.
326          IF(AQAT(1,1).EQ.0.D0) THEN
327             DO 25 J=1,N
328                DF(J)=0.D0
329    25       CONTINUE
330             OK=.FALSE.
331             PRINT *,' setting DF=0 for scalar AQAT=0'
332             RETURN
333          ENDIF
334          DO 16 J=1,N
335             DO 17 JJ=1,N
336                YYT(J,JJ)=(1.D0/AQAT(1,1))*AQ(1,J)*AQ(1,JJ)
337    17       CONTINUE
338    16    CONTINUE
339       ENDIF
```

We need to calculate $QA^T(AQA^T)^{-1}AQ$ for finding d (DF in the code). We know that Q is a symmetric and positive definite matrix and A has full rank m_E, so we can use Choleski factorization to factor AQA^T into upper and lower triangular factors H and H^T respectively. Because MATFAC factors in place, H and H^T are stored in AQAT in the code $\boxed{301}$. The subroutine MATFAC factors AQAT into lower and upper triangular matrices and if the matrix to be factored is not positive definite, its return code is not zero. In the case of linear equality constraints, this return code RC should always be zero, but in practice, roundoff errors sometimes cause the matrix AQAT to be non-positive-definite. In the case of nonlinear equality constraints, as demonstrated in §4.1.3, there is always a chance of the matrix A having one or more rows of zero, thus making the matrix in MATFAC non-positive-definite. When MATFAC returns an RC that is other than zero, the subprogram DIRECT returns OK as false and DF as zero, indicating we cannot make any move from the current point.

Otherwise if the return code RC from MATFAC is zero, then we use the lower triangular matrix from the factorization and find Y by solving N systems of linear equations using LOSOLV. To see how this works, think of factoring $QA^T(AQA^T)^{-1}QA$ into two factors Y and Y^T so that

$$\begin{aligned} YY^T &= QA^T(AQA^T)^{-1}AQ \\ &= QA^T(HH^T)^{-1}AQ \\ &= QA^T(H^T)^{-1}(H^{-1})AQ. \\ \text{Then } Y^T &= H^{-1}AQ \\ \text{so } HY^T &= AQ. \end{aligned}$$

Recall that H is the lower triangle of AQAT. In the code we solve this set of n linear systems $HY_j^T = (AQ)_j$ using LOSOLV $\boxed{309\text{–}311}$. Finding YYT gives us the desired $QA^T(AQA^T)^{-1}QA$.

In case we have only one equality constraint we do not need to use MATFAC or LOSOLV. When ME= 1, A is a matrix with only one row, thus making AQAT a scalar. Here $\text{AQAT}^{-1} = \frac{1}{\text{AQAT}}$ and we find YYT as described in lines $\boxed{334\text{–}338}$. For nonlinear equality constraints, at certain points it is possible for AQAT to be a scalar zero, thus making it impossible to find YYT as its reciprocal. In that case, we return OK as false and set DF to be zero. Once we have YYT, we find $Q - QA^T(AQA^T)^{-1}AQ$ and call it QLOW.

```
340 C
341 C     Find QLOW=Q-YYT
342     3 DO 18 J=1,N
343          DO 19 JJ=1,N
344             QLOW(J,JJ)=Q(J,JJ)-YYT(J,JJ)
345    19    CONTINUE
346    18 CONTINUE
347 C
348 C     Find the D without scaling factor, DF=QLOW*D
349       DO 20 J=1,N
350          DF(J)=0.DO
351          DO 21 JJ=1,N
352             DF(J)=DF(J)+QLOW(J,JJ)*D(JJ)
353    21    CONTINUE
354    20 CONTINUE
355 C
356 C     Find the scaling factor such that DF'*Qinv*DF=1
357 C     First find D'*QLOW*D
358       S=0.DO
359       DO 22 J=1,N
360          S=S+D(J)*DF(J)
361    22 CONTINUE
362       IF(S.LE.0.DO) THEN
363 C     we are done
364          PRINT *,'S=',S,' setting DF=0'
365          DO 23 J=1,N
366             DF(J)=0.DO
367    23    CONTINUE
368          OK=.FALSE.
369       ELSE
370          SF=1.DO/DSQRT(S)
371          DO 24 J=1,N
372             DF(J)=SF*DF(J)
373    24    CONTINUE
374       ENDIF
375       RETURN
376       END
```

Now we can determine $\boldsymbol{d}$ in the new algorithm (DF in the code). Recall from §2.2 that

$$\boldsymbol{d} = -\frac{(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}{\sqrt{\boldsymbol{g}^T(\boldsymbol{Q} - \boldsymbol{Q}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{Q}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{Q})\boldsymbol{g}}}.$$

The denominator in this formula can be considered a scaling factor that makes $\boldsymbol{x}^k + \boldsymbol{d}$ a point in the boundary of the current ellipsoid $\boldsymbol{Q}^k$. In the code this scale factor is called S and found $\boxed{\texttt{358-374}}$ as

$$\mathtt{S} = \frac{1}{\sqrt{\mathtt{D} * \mathtt{QLOW} * \mathtt{D}}}$$

where D in the code is $\boxed{\texttt{148-150}}$ $-g$. We can then write DF in terms of D and QLOW as DF = S * QLOW * D. This completes the calculation of $\boldsymbol{d}$.

For finding scaling factor S, it is necessary that the new QLOW matrix be positive definite. If that is not true, then S comes out to be a nonpositive number and we cannot find the scaling factor SF = $\frac{1}{\sqrt{S}}$, so we have to terminate with DF = 0. Another reason S could be a very small negative number is that the current ellipsoid is so small that there is no room for a move, which means that we are very close to the optimal point. As always it is worth mentioning that for linear equalities in analytical calculation S will never be anything other than a positive number. Nonlinear equalities and roundoff errors are the two reasons why S sometimes is a non-positive number. If S becomes non-positive, we might get a better solution by restarting with a new ellipsoid.

3.4 Projecting Onto a Flat

```
C
Code by Sharmila  Shah
C
      SUBROUTINE PRJECT(X, OK1)
C     This routine projects X onto the flat of the equalities.
C
C     variable  meaning
C     --------  -------
C     AAT       matrix A*AT
C     ALPHA     vector of row coefficients
C     AT        transpose of A
C     F         value of the function
C     FCN       function subprogram
C     GRAD      subprogram gets gradient of I'th function
C     I         index on equalities
C     II        second index on equalities
C     J         index on variables
C     JJ        second index on variables
C     K         index for matrix multiplication
C     LSSOLV    routine solves a linear system
C     MATFAC    routine factors a matrix
C     ME        number of equality constraints
C     MI        number of inequalities
C     N         number of variables in the problem
C     OK1       T => X can be projected.
C     RC        return code from MATFAC; 0 => ok
C     SW3       switches tell what is known in /NGC3/
C     W         b-A*x
C     X         the current ellipsoid center
C
C     formal parameters
      REAL*8 X(50)
      LOGICAL*4 OK1
C
C     receive problem data
      COMMON /NGC3/ SW3,N,MI,ME
      LOGICAL*4 SW3(5)
C
C     local variables
      REAL*8 ALPHA(50),AAT(50,50),F(50),AT(50,50)
      REAL*8 FCN
      INTEGER*4 RC
C
C -----------------------------------------------------------------
C
C     Assume X can be projected.
      OK1=.TRUE.
C
C     if there are no equalities, return the original vector
      IF(ME.EQ.0) RETURN
```

When there are no equality constraints, no projection is required so we return the original vector X.

```
C
C     evaluate A'
      DO 2 I=1,ME
         CALL GRAD(X,N,MI+I, AT(1,I))
    2 CONTINUE
C
C     find AA' to define the linear system
      DO 3 I=1,ME
         DO 4 II=1,ME
            AAT(I,II)=0.DO
            DO 5 K=1,N
               AAT(I,II)=AAT(I,II)+AT(K,I)*AT(K,II)
    5       CONTINUE
    4    CONTINUE
    3 CONTINUE
```

We find the matrix $\boldsymbol{A}$ needed to define the flat F from the nonlinear equality constraints using their first-order Taylor series expansions, but we do not need to explicitly calculate the vector $\boldsymbol{b}$, as explained below. We also calculate $\boldsymbol{AA}^T$ from $\boldsymbol{A}$.

```
C
C     b=Ax-F and we need to solve the linear system
C     AA'*alpha=b-Ax for alpha, so we solve AA'*alpha=-F
      DO 6 I=1,ME
         F(I)=-FCN(X,N,MI+I)
    6 CONTINUE
      IF(ME.GT.1) THEN
         CALL MATFAC(AAT,50,ME,RC)
         IF (RC.NE.0) THEN
            Print *,'AAT not positive-definite, cannot project'
            OK1=.FALSE.
            RETURN
         ENDIF
         CALL LSSOLV(AAT,50,ME,F, ALPHA)
      ELSE
         IF (AAT(1,1).EQ.0) THEN
            Print *,'scalar AAT not positive-definite, cannot project'
            OK1=.FALSE.
            RETURN
         ENDIF
         ALPHA(1)=F(1)/AAT(1,1)
      ENDIF
```

If the vector of equality constraint functions is

$$\boldsymbol{v} = \begin{bmatrix} f_{m_I+1}(\boldsymbol{x}^k) \\ \vdots \\ f_{m_I+m_E}(\boldsymbol{x}^k) \end{bmatrix}$$

then from §2.4, $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{v}$. We know from §2.3 that $\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^k$. Substituting for $\boldsymbol{b}$ we have

$$\begin{aligned} \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} &= \boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{v} - \boldsymbol{A}\boldsymbol{x}^k. \\ \text{Thus, } \boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\alpha} &= -\boldsymbol{v}. \end{aligned}$$

We can solve the above linear system of equations to find $\boldsymbol{\alpha}$. The subroutine MATFAC is used to do the Choleski factorization of the matrix $\boldsymbol{A}\boldsymbol{A}^T$. We know that $\boldsymbol{A}$ is a matrix with maximum possible rank m_E, so $\boldsymbol{A}\boldsymbol{A}^T$ is a symmetric positive definite matrix and we can use the Choleski factorization to split it into lower and upper triangular matrices. If the linearization of nonlinear equalities causes AAT to be a non-positive-definite matrix, then the return code from MATFAC is not zero and we return the original vector without projecting it and return OK1 as false. If the factorization succeeds, the factors stored by in AAT are used by LSSOLV to solve the system of m_E linear equations using forward and back substitution. When we have only one equality constraint, the matrix AAT is in fact only a scalar and we can find ALPHA as described in $\boxed{462}$. We need to make sure this scalar is nonzero as well, otherwise we return OK1 as false.

```
464 C
465 C     find xf=x+A'alpha
466       DO 7 J=1,N
467          DO 8 JJ=1,ME
468             X(J)=X(J)+AT(J,JJ)*ALPHA(JJ)
469     8    CONTINUE
470     7 CONTINUE
471       RETURN
472       END
```

Having found $\boldsymbol{\alpha}$, we find the projection of $\boldsymbol{x}^k$ on the flat F as $\boldsymbol{x}^k_F = \boldsymbol{x}^k + \boldsymbol{A}^T\boldsymbol{\alpha}$. As mentioned in §2.3, we require projection as a precautionary measure for numerical stability for problems with linear equality constraints and as a necessity for problems with nonlinear equalities.

CHAPTER 4
Computational Experience

4.1 Test Problems

To test this algorithm, we used problems from [3], [1], [10] and [11] as summarized in the table on the next page. Out of these 33 test problems, 17 are convex for the inequality-constrained problem. Five out of these 17 problems have nonlinear equality constraints, making them nonconvex. Thus, altogether we have 21 problems that are nonconvex for various reasons. Even though this algorithm is not guaranteed to converge for nonconvex problems, in practice we find that in most cases it does converge (just as the classical ellipsoid algorithm frequently converges on nonconvex problems of full dimension). The other 12 problems are convex inequality-constrained problems with linear equality constraints that make the dimension of their feasible set smaller than n, the dimension of the problem space. In all cases, I used an equality constraint tolerance `T` of 10^{-6} in the program of §3.

Table 4.1 lists the important characteristics of the problems: the number of variables `N`, the number of inequality constraints `MI`, the number of linear equality constraints, the number of nonlinear equality constraints, whether the inequality-constrained problem is convex, and a reference where the problem is published. Each problem in [11] came with information on starting point, and an alleged optimal objective value and optimal point. The problems in [1] are given in a textbook, mostly without information on starting points or optimal solutions. The problems in [10] are accompanied by information about the starting point and alleged optimal points and values. For all of the problems we used from [10], the solutions given in the reference violate some of the equality constraints by as much as 10^{-2}. The solutions we found satisfy the equalities within 10^{-13} in all cases and within 10^{-18} in most cases. The problems `LINEAR` and `JM` were created to test various aspects of this algorithm.

Problem	N	MI	ME = sum of		Inequality Problem	Reference
			Linear	Nonlinear	Convex ?	
HS6	2	0		1	No	[11]
HS7	2	0		1	No	[11]
HS8	2	0		2	No	[11]
HS26	3	0		1	No	[11]
HS28	3	0		1	Yes	[11]
HS39	4	0		2	No	[11]
HS40	4	0		3	No	[11]
HS46	5	0		2	No	[11]
HS48	5	0	2		Yes	[11]
HS49	5	0	2		Yes	[11]
HS50	5	0	3		Yes	[11]
HS51	5	0	3		Yes	[11]
HS52	5	0	3		Yes	[11]
HS107	9	8		6	No	[11]
HS109	9	20		6	No	[11]
HS119	16	32	8		Yes	[11]
LINEAR	2	0	1		Yes	§A
BS400	2	0	1		Yes	[1]
BS401	3	1		1	Yes	[1]
BS403	2	1		1	No	[1]
BS403C	2	1		1	No	[1]
BS467	4	4	2		Yes	[1]
BS475	3	4	1		Yes	[1]
BS475A	2	1	1		Yes	[1]
BS476	3	1		1	No	[1]
BS486	2	1		1	Yes	[1]
JM	3	1	1		Yes	§A
HIM4	10	10	3		No	[10]
HIM4A	10	4		3	No	[10]
HIM5	3	3	1	1	Yes	[10]
HIM15	6	12		4	Yes	[10], §A
HIM20	24	30	2	12	No	[10]
BRM4	10	28		4	No	[3]

Table 4.1: Summary of Test Problems

4.1.1 HS48 - HS52

These five problems from [11] are all convex with linear equality constraints. They are all 5-dimensional problems. These are the type of nonlinear programming problem the new algorithm of §2.5 is guaranteed to solve, and solve them it does! For all of these, the biggest error in satisfying the equalities is on the order of 10^{-15}. In all cases, at least one of the equalities has a value of exactly zero.

4.1.2 HS107

I got a better record value for `HS107` than the one specified in [11], at a feasible point that also satisfies the equality constraints more precisely. For further details refer to tables 4.3 and 4.4.

4.1.3 HS109

In the case of `HS109`, the new algorithm fails using the starting point specified, the origin. The origin is infeasible for both the equality and inequality constraints, thus making it necessary to project it onto a flat of linearized equality constraints. At the origin, the linearization of the equality constraints causes the matrix $\boldsymbol{A}$ to have two zero rows. This in turn makes the matrix $\boldsymbol{AQA}^T$ non-positive-definite. The routine `MATFAC` can work out the Choleski factorization only of positive definite matrices. In this situation it cannot factorize the matrix and we cannot find the projection of the starting point $\boldsymbol{x}^0$ on the linearized flat of equalities, so we cannot solve the problem with the given starting point. From a different starting point the answer I found is better than the one specified in the book in two respects. It gives a better optimal point and the equality constraints are solved more precisely. At the solution specified in the book, the constraint having `I=26` is violated by on the order of 10^3 thus making the solution given in the book grossly infeasible for the equality constraint.

4.1.4 HS119

The reference for `HS119` gives an optimal objective value that is not the value the objective function takes on at their solution point. Even though the objective value specified in the book is lower than the one found by my algorithm, the errors in satisfying the equalities are dramatically larger for their solution. For my answer, the worst error in satisfying an equality constraint is -8.8×10^{-16} whereas the solution point in the reference has a worst error of about -2.3×10^{-1}.

4.1.5 BS403 and BS403C

Graphs for the two-dimensional problems BS403 and BS403C are shown below. Each of these problems has a nonlinear equality constraint that is a circle and one linear inequality constraint; the other lines are the contours of the objective function.

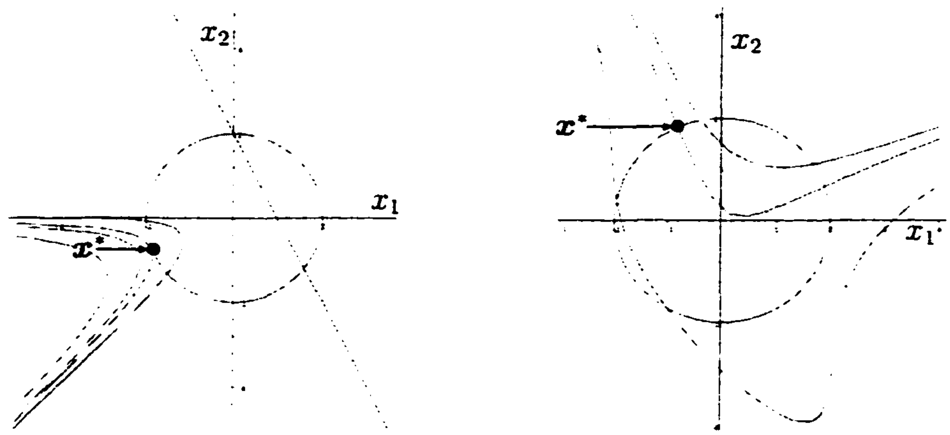


Figure 4.1: BS403 and BS403C

4.1.6 BS476

This problem is non-convex even without the equality constraint and also has a nonlinear equality constraint. The problem is stated without any initial point. Starting from a set of upper and lower bounds deduced from the equality and inequality constraints leads to the optimal point. Using other arbitrary sets of bounds leads to a point that is feasible for both equality and inequality constraints but is not an optimal point.

4.1.7 HIM15

The reference for HIM15 gives an optimal objective value that is not the value the objective function takes on at their solution point. This problem has a differential equation for its objective function. To solve this problem, we solved the differential equation and used the resulting formula for the objective function. Even though the solution specified in the book has an objective value lower than the one found by our algorithm, the errors in satisfying the equalities are dramatically larger for their solution. For our answer, the worst error in satisfying an equality constraint is -5.7×10^{-14} whereas the solution point in the reference has a worst error of about -3.5×10^{-1} for the I= 14 constraint. The objective function in the reference [10] for this problem and the objective function we used are specified in §A.

4.1.8 BRM4

The problem BRM4, the alkylation process problem, also appears as HS117 in [11]. For this problem, the solution given in both the references (which is the same) is infeasible for the inequality constraint having I=27 and the worst error for satisfying an equality constraint is -1.1777. The answer we find is feasible for all inequalities and the worst error in satisfying an equality constraint is 4.54×10^{-13}. Also the optimal objective value given in the reference, while slightly smaller than our objective value, is not the value the objective function takes on at their solution point.

4.2 Experimental Procedure

As discussed in §2.5, we use the bounds on the variables to find the starting ellipsoid, the center of which is the midpoint of the bounds in each direction. I tried recentering for all the problems and in all cases it improved the answers. This was true most conspicuously for problems with nonlinear equality constraints because the Taylor series approximation is highly dependent on the specific nonlinear equality and also on the point at which the linearization is done.

No specific tolerance for termination was defined and I ran the code until the answers stopped changing. For those problems where the starting point was not defined, I tried several starting points to see which one gave the best answer. The problems from [1] except BS467 and BS475A had no starting point specified for them. The following table lists the starting points we used for those problems and for the problems LINEAR, JM, and HS109.

Problem	Starting Point		
BS400	2.00000000D+00,	2.00000000D+00	
BS401	-2.50000000D+00,	2.00000000D+00,	2.00000000D+00
BS403	1.00000000D+00,	3.00000000D+00	
BS403C	-1.00000000D+00,	0.00000000D+00	
BS475	0.00000000D+00,	1.50000000D+00,	5.00000000D-01
BS486	1.50000000D+00,	1.50000000D+00	
LINEAR	1.00000000D+00,	0.00000000D+00	
JM	0.00000000D+00,	0.00000000D+00,	1.00000000D+00
HS109	3.68437721D+02,	1.62399321D+03,	1.00000000D+00,
	1.00000000D+00,	2.24000000D+02,	2.24000000D+02,
	2.24000000D+02,	2.00000000D+02,	2.00000000D+02

Table 4.2: Starting Points

As in any other algorithm for nonlinear optimization, performance depends significantly on the starting point choices. For more than half of these problems, the starting point I used was infeasible for the equality constraints. Recall from §2.3 that a starting point that is not feasible for the equalities is first projected on the flat of linearized equality constraints.

4.3 Results and Discussion

The following table contains a summary of the results obtained by the new algorithm. When it says that the algorithm solved a particular problem, the solution is strictly feasible for all inequality constraints, it also satisfies the equality constraints within tolerance of 10^{-6}, and it has a record value that does not change even after recentering repeatedly. As mentioned in §4.1, for some of the problems our answers were much closer to the set of equalities than the answers published in the reference. When no optimal value was reported, or when our optimal value is better than the one reported, or when the previously reported optimal value is at a grossly infeasible point, this fact is indicated in the table of experimental results by a smiley face ☺. Thus when ☺ appears, it indicates that we have a much better optimal point. For example, for HS107 our record point and the record point in the book are as follows.

x Component	Our Record Point	Record Point in [11]
x_1	6.6701058647298539D-01	6.667095D-01
x_2	1.0223870233541728D+00	1.022388D+00
x_3	2.2828766345226159D-01	2.282879D-01
x_4	1.8482174827168008D-01	1.848217D-01
x_5	1.0908999935263124D+00	1.090900D+00
x_6	1.0908999990292667D+00	1.090900D+00
x_7	1.0690359513811059D+00	1.069036D+00
x_8	1.0661197142591200D-01	1.066126D+00
x_9	-3.3878698073970215D-01	-3.387867D-01

Table 4.3: Comparison of Results for HS107

The values of the equality constraints at our record point and the one in the reference are listed in the following table.

Equality Constraint	Value at Our Record Point	Value at Record Point in [11]
f_9	5.5511151231257827D-17	-7.0985807553625468D-01
f_{10}	-1.1102230246251565D-16	1.8109364232211713D+00
f_{11}	3.3306690738754696D-16	-4.0021992172954635D-01
f_{12}	0.0000000000000000D+00	7.8652067549629057D-01
f_{13}	-2.2204460492503131D-16	9.9881933110282017D-01
f_{14}	1.4432899320127035D-15	9.5823501901140129D-01

Table 4.4: Comparison of Equalities for HS107

Problem	N MI ME	Largest Equality Error	Our Record Value	Source Objective Value
HS6	2 0 (1)	0.00000000D+00	2.52939621D-23	0.00000000D+00
HS7	2 0 (1)	0.00000000D+00	-1.73205080D+00	-1.73205080D+00
HS8	2 0 (2)	0.00000000D+00	-1.00000000D+00	-1.00000000D+00
HS26	3 0 (1)	0.00000000D+00	2.21836075D-17	0.00000000D+00
HS28	3 0 (1)	-2.22044605D-16	4.70378228D-20	0.00000000D+00
HS39	4 0 (2)	4.03016157D-17	-9.99999999D-01	-1.00000000+D00
HS40	4 0 (3)	9.54791801D-15	-2.49999999D-01	-2.50000000D-01
HS46	5 0 (2)	-1.11022302D-16	2.35897909D-20	0.00000000D+00
HS48	5 0 2	4.44089210D-16	1.18982209D-13	0.00000000D+00
HS49	5 0 2	0.00000000D+00	2.86197143D-23	0.00000000D+00
HS50	5 0 3	-1.77635684D-15	2.28943097D-21	0.00000000D+00
HS51	5 0 3	8.88178420D-16	9.25074021D-23	0.00000000D+00
HS52	5 0 3	4.16333634D-17	5.32664756D+00 ☺	5.32664764D+00
HS107	9 8 (6)	1.44328993D-15	5.05501145D+03 ☺	5.05501180D+03
HS109	9 20 (6)	3.93356458D-11	5.32685133D+03 ☺	5.36206928D+03
HS119	16 32 8	-8.88178419D-16	1.31363621D+02 ☺	1.32850466D+02
LINEAR	2 0 1	0.00000000D+00	7.50000000D-01	7.50000000D-01
BS400	2 0 1	-8.88178419D-16	3.35528345D+01	3.35528345D+01
BS401	3 1 (1)	-2.83995049D-13	-6.82207465D+00 ☺	none stated
BS403	2 1 (1)	5.32907051D-15	-6.71459293D-01 ☺	none stated
BS403C	2 1 (1)	0.00000000D+00	7.22128142D-01	7.22128128D-01
BS467	4 4 2	-8.88178420D-16	-7.16129029D+00	-7.16129032D+00
BS475	3 4 1	0.00000000D+00	-2.39999999D+01	-2.40000000D+01
BS475A	2 1 1	0.00000000D+00	1.00000000D+00	1.00000000D+00
BS476	3 1 (1)	0.00000000D+00	-2.01416753D+01	-2.01416753D+01
BS486	2 1 (1)	0.00000000D+00	1.20000000D+01	1.20000000D+01
JM	3 1 1	0.00000000D+00	-1.50000000D+00	-1.50000000D+00
HIM4	10 10 3	2.22044604D-16	-4.77610909D+01	-4.77699809D+01
HIM4A	10 4 (3)	2.22044604D-15	-4.77610909D+01	-4.77699809D+01
HIM5	3 3 (2)	0.00000000D+00	9.61715172D+02 ☺	9.61718246D+02
HIM15	6 12 (4)	1.27897692D-13	8.82759774D+03 ☺	8.82758000D+03
HIM20	24 30 (14)	2.22044604D-16	5.17277731D-02 ☺	5.70595712D-02
BRM4	10 28 (3)	4.54747351D-13	-1.16133660D+03 ☺	-1.76880696D+03

Table 4.5: Experimental Results

CHAPTER 5
Conclusions, and Future Work

5.1 Conclusions from this Work

The new algorithm solves convex problems with linear equality constraints, with or without inequality constraints, starting from points that are feasible or infeasible for the equalities. In addition, it solves some otherwise convex problems having nonlinear equality constraints, and it solved most of the problems we tried that are nonconvex even ignoring the equalities.

For numerical stability of the algorithm, it is necessary to re-project the center found at each iteration onto the flat of equalities.

The accuracy of this algorithm is greatly improved by recentering.

5.2 Directions for Future Work

More systematic computational testing is needed to further evaluate the performance of this algorithm. Timing measurements on a larger set of test problems could be used to compare the new method with other available algorithms for nonlinear programs having both equality and inequality constraints.

For serious computational tasks, the algorithm should be re-implemented in production code.

When we start from a point that is infeasible for the equalities, we project it onto the flat of equalities. This scheme works very well for linear equalities. For problems with nonlinear equalities, if the starting point or any subsequent points are not strictly feasible for the equality constraints, then the Taylor series approximation that we use to linearize the surface given by the equalities may be far from accurate. So, I want to improve robustness for nonlinear equalities.

The new algorithm can be used to solve systems of nonlinear equations, so it might be interesting to experiment with some problems of that sort.

LITERATURE CITED

[1] M. S. Bazaraa, H. D. Sherali and C. M. Shetti, "*Nonlinear Programming Theory and Algorithms*," John Wiley & Sons, New York, 1993.

[2] R. G. Bland, D. Goldfarb and M. J. Todd, "The Ellipsoid Method: a Survey," *Operations Research* 29 (1981) 1039-1091.

[3] J. Bracken and G. P. McCormick, "*Selected Applications of Nonlinear Programming*," John Wiley & Sons, New York, 1968.

[4] J. G. Ecker and M. Kupferschmid, "An Ellipsoid Algorithm for Nonlinear Programming," *Mathematical Programming* 27 (1983) 83-106.

[5] J. G. Ecker and M. Kupferschmid, "A Computational Comparison of the Ellipsoid Algorithm With Several Nonlinear Programming Algorithms," *SIAM Journal on Control and Optimization* 23 (1985) 657-674.

[6] Shu-Cherng Fang and S. Puthenpura, "*Linear Optimization and Extensions: Theory and Algorithms*," Prentice Hall, New Jersey, 1993.

[7] H. B. Fine and H. D. Thompson, "*Coordinate Geometry*," The Macmillan Company, New York, 1930.

[8] Jean-Louis Goffin, "Convergence Rates of the Ellipsoid Method on General Convex Functions," *Mathematics of Operations Research* 8 (1983) 135-150.

[9] M. Grötschel, L. Lovász, A. Schrijver, "*Geometric Algorithms and Combinatorial Optimization*," Springer-Verlag, New York, 1985.

[10] D. M. Himmelblau, "*Applied Nonlinear Programming*," McGraw-Hill, New York, 1972.

[11] W. Hock and K. Schitkowski, "Test Examples For Nonlinear Programming," in *Lecture Notes in Economics and Mathematical Systems*, Vol. 187, Springer-Verlag, New York, 1981.

[12] Fritz John, "Extremum Problems with Inequalities as Subsidiary Conditions," in *Studies and Essays, presented to R. Courant on his 60th Birthday, January 8, 1948*, Interscience, New York (1948) 187-204 (Reprinted in: J. Moser (ed.) *Fritz John, Collected Papers Volume 2*, Birkhäuser (1985) 543-560).

[13] L. G. Khachian, "A Polynomial Algorithm in Linear Programming," *Soviet Mathematics Doklady* 20 (1979) 191-194.

[14] M. Kupferschmid, "*An Ellipsoid Algorithm for Convex Programming,*" Ph.D. Dissertation, Rensselaer Polytechnic Institute, Troy, NY, 1981.

[15] Hans-Jakob Lüthi, "On the Solution of Variational Inequalities by the Ellipsoid Method," *Mathematics of Operations Research* 10 (1985) 515-522.

[16] N. Z. Shor, "Cut-off Method With Space Extension in Convex Programming Problems," *Cybernetics* 12 (1977) 94-96.

[17] Gilbert Strang, "*Linear Algebra and Its Applications,*" Academic Press, New York, 1976.

[18] D. B. Yudin and A. S. Nemirovskii, "Informational Complexity and Effective Methods for Solving Convex Extremal Problems," *Matekon*, 13 (1977) 24-45.

APPENDIX A
Some Specific Problems

- Problem `Linear`

This is one of the problems I made up to test the algorithm. It is a very simple small problem with no inequality constraints. I used it to illustrate the new algorithm in §2.1.

$$
\begin{aligned}
\min_{x \in \Re^2} \quad & 3x_1^2 + x_2^2 \\
\text{subject to} \quad x_1 + x_2 - 1 &= 0
\end{aligned}
$$

The optimal value for this problem is 0.75 and the optimal point is $(0.25, 0.75)^T$.

- Problem `JM`

This is the second problem I made up to test the new algorithm. This problem helped reveal why the two-dimensional approach for solving the equality constrained problems would not work for higher dimensions. It shows that just taking the projection of $\boldsymbol{g}$ on F and scaling it so that the $\boldsymbol{d}$ lies in the boundary of the ellipsoid is not sufficient. If we found the $\boldsymbol{d}$ in the way described in §2.1, this $\boldsymbol{d}$ will produce a hyperplane that might cut off the half of the ellipsoid that contains the optimal point.

$$
\begin{aligned}
\min_{x \in \Re^3} \quad & x_1 - x_2 - x_3 \\
\text{subject to} \quad (x_1 + 2.5)^2 + (x_2)^2 - 8 &\leq 0 \\
x_3 &= 0
\end{aligned}
$$

The optimal value for this problem is -1.5, with $\boldsymbol{x}^* = (-0.5, 2.0, 0)^T$.

- Problem HIM15

This is the objective function for the problem HIM15 as given in the reference [10].

$$
\begin{aligned}
\text{Minimize } f(\boldsymbol{x}) &= f_1(x_1) + f_2(x_2) \\
\text{where } f_1(0) &= 0, \quad f_2(0) = 0 \\
\frac{df_1}{dx_1} &= \begin{cases} 30 & \text{if} \quad 0 \le x_1 < 300 \\ 31 & \text{if} \quad 300 \le x_1 \le 400 \end{cases} \\
\frac{df_2}{dx_2} &= \begin{cases} 28 & \text{if} \quad 0 \le x_2 < 100 \\ 29 & \text{if} \quad 100 \le x_2 < 200 \\ 30 & \text{if} \quad 200 \le x_2 < 300 \end{cases}
\end{aligned}
$$

We used the following objective function for the problem we solved.

$$
\begin{aligned}
\text{Minimize } f(\boldsymbol{x}) &= f_1(x_1) + f_2(x_2) \\
f_1(x_1) &= \begin{cases} 30x_1 & \text{if} \quad 0 \le x_1 < 300 \\ 31x_1 - 300 & \text{if} \quad 300 \le x_1 \end{cases} \\
f_2(x_2) &= \begin{cases} 28x_2 & \text{if} \quad 0 \le x_2 < 100 \\ 29x_2 - 100 & \text{if} \quad 100 \le x_2 < 200 \\ 30x_2 - 300 & \text{if} \quad 200 \le x_2 \end{cases}
\end{aligned}
$$

IMAGE EVALUATION TEST TARGET (QA-3)

150mm

6"

APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989