ARC SEARCH METHODS FOR LINEARLY CONSTRAINED OPTIMIZATION

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR
COMPUTATIONAL AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Nicholas Wayne Henderson
June 2012

Abstract

We present a general arc search algorithm for linearly constrained optimization. The method constructs and searches along smooth arcs that satisfy a small and practical set of properties. An active-set strategy is used to manage linear inequality constraints. When second derivatives are used, the method is shown to converge to a second-order critical point and have a quadratic rate of convergence under standard conditions. The theory is applied to the methods of line search, curvilinear search, and modified gradient flow that have previously been proposed for unconstrained problems. A key issue when generalizing unconstrained methods to linearly constrained problems using an active-set strategy is the complexity of how the arc intersects hyperplanes. We introduce a new arc that is derived from the regularized Newton equation. Computing the intersection between this arc and a linear constraint reduces to finding the roots of a quadratic polynomial. The new arc scales to large problems, does not require modification to the Hessian, and is rarely dependent on the scaling of directions of negative curvature. Numerical experiments show the effectiveness of this arc search method on problems from the CUTEr test set and on a specific class of problems for which identifying negative curvature is critical. A second set of experiments demonstrates that when using SR1 quasi-Newton updates, this arc search method is competitive with a line search method using BFGS updates.

Acknowledgments

I've had the great privilege of knowing and working with many wonderful people during my time at Stanford University.

My principal advisor, Walter Murray, has been a great mentor and friend. He helped me land my first internship the summer before I started at Stanford. He put me forward for various fellowships, which supported this research. His class on numerical optimization initiated my interest in the field. I will always value his guidance and advice.

Michael Saunders has also been a great supporter and friend. Any reader of this dissertation who is unable to find a dangling participle should thank him. Michael is kind of heart and extremely generous with his time. His excellent class on large-scale linear algebra and optimization was instrumental in the development of my solver.

My other committee members were Yinyu Ye, Margot Gerritsen, and Robert Tibshirani. They are exemplary teachers, researchers, and people.

ICME has been a wonderful home during my time at Stanford. I would like to thank the past and present directors Peter Glynn, Walter Murray, and Margot Gerritsen for their dedication to the program. Indira Choudhury deserves special credit for her tireless support of students.

Finally, I would like to express my gratitude to my family and friends. I enjoy life because they are a part of it.

This research was supported by the William R. Hewlett Stanford Graduate Fellowship Fund and grants from the Office of Naval Research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preliminaries

This thesis is concerned with algorithms to find local solutions to the linearly constrained optimization problem

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & Ax \geq b. \end{array} \tag{LC}$$

More specifically, we seek points that satisfy the first- and second-order necessary conditions for a minimizer. Here, $F \in C^2 : \mathbb{R}^n \mapsto \mathbb{R}$, $A \in \mathbb{R}^{m\times n}$, and $b \in \mathbb{R}^m$. In general, it is assumed the gradient $\nabla F(x)$ and the Hessian $\nabla^2 F(x)$ are available. It will be seen that $\nabla^2 F(x)$ is only needed in the form of an operator for matrix-vector products.

Algorithms for this problem usually generate a sequence of iterates $\{x_k\}_{k=0}^{\infty}$, which converge to a solution x^*. At each iterate, the gradient is denoted $g_k = \nabla F(x_k)$ and the Hessian is $H_k = \nabla^2 F(x_k)$. Arc search methods produce new iterates with the update

$$x_{k+1} = x_k + \Gamma_k(\alpha_k),$$

where $\Gamma_k \in C^2 : \mathbb{R} \mapsto \mathbb{R}^n$ and $\alpha_k \geq 0$. $\Gamma_k(\alpha)$ is a smooth arc in n-dimensional Euclidean space and α is the step size. For $\alpha = 0$ arcs have no displacement ($\Gamma_k(0) = 0$) and are initially tangent to a descent direction ($\Gamma_k'(0)^T g_k < 0$). The second condition may be relaxed to $\Gamma_k'(0)^T g_k \leq 0$ if a direction of negative curvature is used.

In the thesis we prove global convergence for a generic arc search algorithm to solve LC. An active-set strategy is used to manage the linear inequality constraints. When second derivatives are used we show convergence to second-order critical points and a quadratic rate of convergence under standard conditions. We also develop a specific arc that scales to large problems, does not require Hessian modification, and avoids arbitrary scaling choices. This chapter discusses arc search methods in the context of the well known line search and trust-region methods.

1.2 Unconstrained optimization

Algorithms for LC typically are extensions of methods to solve the unconstrained optimization problem,

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad F(x). \tag{UC}$$

The best possible solution to UC is called a global minimizer, a point in $\mathbb{R}^n$ that attains the least value of F.

Definition 1.1. *If $F(x^*) \leq F(x)$ for all x, then x^* is called a global minimizer.*

Unfortunately, global minimizers can be very difficult to find. Thus, we restrict our interest to methods that find local minimizers. Note that many methods to find global minimizers are based on methods to find local minimizers.

Definition 1.2. *If $F(x^*) \leq F(x)$ for all x in some neighborhood of x^*, then x^* is called a local minimizer.*

Definition 1.3. *If $F(x^*) < F(x)$ for all x in some neighborhood of x^* with $x^* \neq x$, then x^* is called a strict local minimizer.*

Definitions 1.2 and 1.3 are not possible to check computationally. When F is smooth, derivatives can be used to derive optimality conditions.

Theorem 1.1. *If x^* is a local minimizer then*

$$\nabla F(x^*) = 0 \text{ and } \nabla^2 F(x^*) \succeq 0. \tag{1.1}$$

Theorem 1.2. *x^* is a strict local minimizer if*

$$\nabla F(x^*) = 0 \text{ and } \nabla^2 F(x^*) \succ 0. \tag{1.2}$$

Equations (1.1) are known as the second-order *necessary* optimality conditions. If F is bounded below on a compact set, then a point satisfying (1.1) exists and is called a second-order *critical* point. Equations (1.2) are known as the second-order *sufficient* optimality conditions. Points satisfying (1.2) need not exist. For example, $x^* = 0$ is a strict (global) minimizer for $F(x) = x^4$, yet $F''(0) = 0$. Thus algorithms can usually only guarantee the ability to find points satisfying (1.1) and in practice often find points satisfying (1.2).

1.3 Linearly constrained optimization

Consider now problems that are constrained by a set of linear inequalities, $Ax \geq b$. Here, A is a $m \times n$ matrix and b is a vector of length m. An individual constraint is written $a_i^T x \geq b_i$, where a_i^T is the ith row of A and b_i is the ith element of b. For a point x, a constraint is said to be *active* if $a_i^T x = b_i$, *inactive* if $a_i^T x > b_i$, and *violated* if $a_i^T x < b_i$.

Solutions and optimality conditions for LC are defined in an analogous fashion to those for UC.

Definition 1.4. *If $Ax^* \geq b$ and there is a neighborhood $\mathcal{N}$ such that $F(x^*) \leq F(x)$ for all $x \in \mathcal{N}$ and $Ax \geq b$ then x^* is called a local minimizer.*

Definition 1.5. *If $Ax^* \geq b$ and there is a neighborhood $\mathcal{N}$ such that $F(x^*) < F(x)$ for all $x \in \mathcal{N}$ and $Ax \geq b$ with $x^* \neq x$ then x^* is called a strict local minimizer.*

Theorem 1.3. *Given a point $x^* \in \mathbb{R}^n$, let $\mathcal{A}$ be the matrix of constraints active at x^* and let Z denote a basis for* null$(\mathcal{A})$. *Then x^* is a local solution to LC only if*

$$\left.\begin{aligned} Ax^* \geq b \text{ with } \mathcal{A}x^* = b \\ \nabla F(x^*) = \mathcal{A}^T\lambda^* \Leftrightarrow Z^T\nabla F(x^*) = 0 \\ \lambda^* \geq 0 \end{aligned}\right\} \quad \textit{(first-order)} \tag{1.3}$$

$$Z^T\nabla^2 F(x^*)Z \succeq 0. \quad \textit{(second-order)} \tag{1.4}$$

Theorem 1.4. *Given a point $x^* \in \mathbb{R}^n$, let $\mathcal{A}$ be the matrix of constraints active at x^* and let Z denote a basis for* null$(\mathcal{A})$. *Then x^* is a strict local solution to LC if*

$$\left.\begin{aligned} Ax^* \geq b \text{ with } \mathcal{A}x^* = b \\ \nabla F(x^*) = \mathcal{A}^T\lambda^* \Leftrightarrow Z^T\nabla F(x^*) = 0 \\ \lambda^* > 0 \end{aligned}\right\} \quad \textit{(first-order)} \tag{1.5}$$

$$Z^T\nabla^2 F(x^*)Z \succ 0. \quad \textit{(second-order)} \tag{1.6}$$

In Theorems 1.3 and 1.4 $Z^T\nabla F(x^*)$ is the *reduced gradient*, $Z^T\nabla^2 F(x^*)Z$ is the *reduced Hessian*, and λ^* is the vector of *Lagrange multipliers*. Combined, equations (1.3) and (1.4) are known as the second-order *necessary* optimality conditions for LC. A second-order *critical* point satisfying (1.3) and (1.4) exists if F is bounded below on a compact set that has a nontrivial intersection with the feasible region defined by $Ax \geq b$. Equations (1.5) and (1.6) are known as the second-order *sufficient* optimality conditions. Points satisfying (1.5) and (1.6) need not exist, just like the case for UC.

1.4 Second derivative methods

1.4.1 Newton

Newton's method is the gold standard in optimization. In the context of unconstrained optimization the algorithm is simply

$$x_{k+1} = x_k - H_k^{-1}g_k. \tag{1.7}$$

In the case where $H_k \succ 0$, x_{k+1} minimizes $m_k(x) = \frac{1}{2}x^T H_k x + g_k^T x$, a local quadratic model of F around x_k. Newton's method has the desirable property that if it converges to a point x^* with

$\nabla^2 F(x^*) \succ 0$, then it does so at a quadratic rate [4,53]. The problem with Newton's method is that it may cycle, diverge, or converge to a maximizer. Also, if H_k is singular, x_{k+1} is not defined.

Many successful algorithms use Newton's iteration when it works and do something different when it does not. One key feature of these algorithms is that they enforce a *descent property*. That is each new iterate must produce a lower objective function value, i.e. $F(x_{k+1}) < F(x_k)$ for $k > 0$. The two common algorithm classes are line search methods and trust-region methods.

1.4.2 Line search and extensions

Line search methods first compute a descent direction p_k and then invoke a univariate search procedure to compute a step size α_k, which ensures $F(x_k + \alpha_k p_k) < F(x_k)$. The overall update is $x_{k+1} = x_k + \alpha_k p_k$.

Newton's method is attained when p_k solves $H_k p_k = -g_k$ and $\alpha_k = 1$. The key issue with line search based Newton methods is dealing with H_k when it is nearly singular or indefinite. It is possible to compute an approximation $\bar{H}_k$, which is sufficiently positive definite. The search direction is then selected by solving $\bar{H}_k s_k = -g_k$. The process to determine $\bar{H}_k$ may also give a direction of negative curvature d_k such that $d_k^T H_k d_k < 0$.

Methods to adapt line search to obtain convergence to second-order critical points started with McCormick's modification of the Armijo step size rule [47]. McCormick's backtracking rule is

$$x_{k+1} = x_k + s_k 2^{-i} + d_k 2^{-i/2},$$

where s_k is a descent direction ($g_k^T s_k < 0$) and d_k is a direction of negative curvature. The backtracking index i starts at 0 and increments by 1 until an acceptable point is found. Moré and Sorensen [48] developed this into a search based update with

$$x_{k+1} = x_k + \alpha_k^2 s_k + \alpha_k d_k, \tag{1.8}$$

which allows for a more sophisticated univariate search procedure to select an acceptable α_k. Goldfarb proved convergence with the update

$$x_{k+1} = x_k + \alpha_k s_k + \alpha_k^2 d_k. \tag{1.9}$$

The basic idea here is that when α_k is small, the update is dominated by s_k, a direction that provides a greater decrease in F when close to x_k. When α_k is large, d_k dominates, which may provide greater decrease in F far from x_k. A downside of (1.9) is that it requires specialized search conditions. Updates (1.8) and (1.9) are members of a class of *curvilinear search* methods, which use low-order polynomial combinations of vectors.

Forsgren and Murray proved second-order convergence with the line search update

$$x_{k+1} = x_k + \alpha_k(s_k + d_k), \tag{1.10}$$

first in the context of problems with linear equality constraints [27] and then with linear inequality constraints [28]. Gould proved convergence using an "adaptive" line search approach [39], which uses a condition to choose between s_k and d_k. Line search is performed along the selected vector for an acceptable step size. Ferris, Lucidi, and Roma [24] used (1.8) to develop a method for unconstrained problems in the nonmonotone framework of Grippo, Lampariello, and Lucidi [41].

All of the methods discussed need to choose a descent direction s_k and a direction of negative curvature d_k. When H_k is indefinite, the best choice for s_k is not clear. Some methods choose to solve

$$(H_k + \gamma_k I)s_k = -g_k$$

where γ_k is selected such that $H_k + \gamma_k I$ is sufficiently positive definite. Modified Cholesky methods [33] solve

$$(H_k + E_k)s_k = -g_k,$$

where E_k is diagonal and constructed to make $H_k + E_k$ sufficiently positive definite in a single factorization. Fang and O'Leary have cataloged many different modified Cholesky algorithms in [23]. Auslender [2] presents some more ways to compute s_k for use in curvilinear search.

Directions of negative curvature provide another difficulty. First, we've already seen four different ways to use d_k in a search-based algorithm. Many authors use (1.8). However, the best choice is not clear. Second, a direction of negative curvature d_k does not have a natural scale with respect to a descent direction s_k. It is possible to come up with many ways to scale d_k, but we don't have a theoretical measure of quality. Gould's adaptive line search method [39] is able to avoid the relative scaling issue by only using one vector at a time.

1.4.3 Gradient flow

Another interesting approach to optimization is based on systems of ordinary differential equations and has been explored by many authors [1, 3, 5–7, 11, 31]. See Behrman's PhD thesis [3, p. 6] for a summary of the history. The most practical methods, as explored by Botsaris, Behrman, and Del Gatto [3, 6, 31], are based on the system of differential equations

$$\begin{aligned} \frac{d}{dt}x(t) &= -\nabla F(x(t)) \\ x(0) &= x_0. \end{aligned} \tag{1.11}$$

If $\nabla F(x(t))$ linearized about x_k and $x(t)$ shifted such that $x(t) = x_k + w_k(t)$, then (1.11) becomes

$$\begin{aligned} \frac{d}{dt}w_k(t) &= -H_k w_k(t) - g_k \\ w_k(0) &= 0. \end{aligned} \tag{1.12}$$

The linear ODE (1.12) can be solved analytically with the spectral decomposition of H_k. The resulting algorithm for unconstrained problems is

$$x_{k+1} = x_k + w_k(t_k),$$

where t_k is selected with a univariate search procedure to satisfy a descent property. This method is able to handle second derivatives naturally. When $H_k \succ 0$, $w_k(t)$ terminates at the Newton step $p_k = -H_k^{-1} g_k$. When H_k is indefinite the arc produced by the ODE is unbounded and will diverge away from a saddle point (in most cases). A key benefit of this method is that it does not require Hessian modification in the indefinite case.

Botsaris and Jacobson introduced the basic idea, but used a modification so that w_k is bounded if H_k is nonsingular [6]. Their proof of first-order convergence requires t_k to be a minimizer of $F(x_k + w_k(t))$. Behrman provided a convergence result with the unmodified solution to (1.12) and practical search conditions to select t_k [3]. Behrman also showed that the method could scale to large problems by projecting the linear ODE onto the space spanned by a small number of Lanczos vectors of H_k. Del Gatto further improved the practicality by proving convergence in the case where the ODE is projected onto a two-dimensional subspace [31]. We discuss the details and address second-order convergence of this method in Chapter 3.

Botsaris also considered an active-set method for problems with linear inequality constraints [9] and a generalized reduced-gradient method for nonlinear equality constraints [8]. However, the methods presented are not applicable to large-scale problems and second-order convergence is not considered.

1.4.4 Trust-region

Trust-region algorithms differ from search-based methods by first choosing a maximum allowable step size, then computing the direction. The trust-region radius is denoted Δ_k and may be modified at different parts of the algorithm. A step s_k is then determined to satisfy $\|s_k\| \le \Delta_k$, where $\|\cdot\|$ is usually the 2-norm or ∞-norm. If $x_k + s_k$ is deemed acceptable by a descent property, then the update $x_{k+1} = x_k + s_k$ is performed. If $x_k + s_k$ is not acceptable, i.e. $F(x_k + s_k) \ge F(x_k)$, then Δ_k is decreased and s_k is recomputed. Trust-region algorithms must also specify a rule for increasing Δ_k under certain conditions on $F(x_k + s_k)$.

One common way to compute s_k is to minimize a quadratic model of F about x_k subject to the 2-norm trust-region constraint. The optimization problem is

$$\begin{array}{ll} \underset{s}{\text{minimize}} & m_k(s) = \frac{1}{2} s^T H_k s + g_k^T s \\ \text{subject to} & s^T s \le \Delta_k. \end{array} \tag{1.13}$$

A key benefit of the trust-region approach is that the subproblem (1.13) is well defined when H_k is singular or indefinite. It is possible to compute global minimizers of (1.13) at the expense of multiple linear system solves. Such expense is impractical for large problems, so many methods choose to approximately solve (1.13). Steihaug presented a practical algorithm based on applying the conjugate gradient method to $H_k s = -g_k$ [56]. Byrd, Schnabel, and Shultz showed that the trust-region subproblem can be solved on a two-dimensional subspace and still maintain the overall convergence properties [14].

In trust-region terminology a *Cauchy point* is a minimizer of the model function along $-g_k$, subject to the trust-region constraint. Global convergence to stationary points is attained if the reduction in the model function by s_k is at least as good as the reduction provided by a Cauchy point. Second-order convergence for trust-region algorithms was proved by Sorensen using the global minimizer of the quadratic model [55]. Shultz provides a proof that relaxes the condition to the computation of what Conn et al. call an *eigenpoint* [16,54]. An eigenpoint is defined as the minimizer of the model function along a vector that has a sufficient component in space spanned by eigenvectors corresponding to negative eigenvalues of H_k. As long as s_k reduces the model function as much as an eigenpoint, then second-order convergence can be obtained. Later, Zhang and Xu demonstrated second-order convergence with an indefinite dogleg path [61].

Trust-region methods can be extended to constrained problems in many ways. Gay presented an approach for linearly constrained optimization [32]. Branch, Coleman, and Li developed a second-order subspace method for problems with simple bounds on the variables [10]. For problems with nonlinear constraints we refer to work by Conn, Gould, Orban, and Toint [15] and also Tseng [57]. The book *Trust-Region Methods* by Conn, Gould, and Toint [16] provides exhaustive coverage of the field.

1.5 Thesis outline

Chapter 2 presents a second-order convergence proof for a general arc search method using an active-set strategy for linearly constrained optimization. The proof generalizes the work by Forsgren and Murray on a line search method in [28]. For convergence, arcs must satisfy a set of properties similar to the properties of sufficient descent and negative curvature for certain vectors in line search methods. A key difference is that an arc may intersect a linear constraint an arbitrary number of times. This means that an arc can move from and be restricted by the same linear constraint in a single iteration, which is not possible along a line. We show that on any subsequence of iterations where this occurs, the multiplier associated with the constraint must converge to a positive value. This observation and a modification to the rules for constraint deletion allow the proof of [28] to be generalized for arcs.

Chapter 3 shows the application of the convergence theory to several different arcs. Forsgren and Murray line search (1.10) as well as Moré and Sorensen curvilinear search (1.8) directly satisfy the sufficient arc conditions with appropriate choices for s_k and d_k. Goldfarb's method (1.9) requires a simple modification to guarantee second-order convergence. We introduce an arc based on the regularized Newton equation and designate it with NEM. The theory is also applied to the modified gradient flow algorithm of Behrman and Del Gatto. All methods may be used on linearly constrained problems. We discuss the derivation and application of NEM arcs in some detail.

Chapter 4 details ARCOPT, a Matlab implementation using NEM arcs for linearly constrained optimization. Chapter 5 covers several numerical experiments with ARCOPT:

- a comparison of ARCOPT and IPOPT on a continuous formulation of the Hamiltonian cycle problem,
- a comparison of ARCOPT, IPOPT, and SNOPT on problems in the CUTEr test set,
- a comparison of quasi-Newton updates with an arc search method on problems in the CUTEr test set.

Chapter 2

Convergence

2.1 Preliminaries

This chapter presents a *general* arc search algorithm and associated convergence theory for the problem

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & Ax \geq b. \end{array}$$

Here A is a real $m \times n$ matrix and $F \in C^2 : \mathbb{R}^n \mapsto \mathbb{R}$. We are interested in algorithms that converge to points satisfying the second-order necessary optimality conditions.

The notation and arguments in this chapter are inspired by and adapted from the work by Forsgren and Murray [27, 28].

2.2 Statement of assumptions

Assumption 2.1. *The objective function F is twice continuously differentiable.*

Assumption 2.2. *The initial feasible point x_0 is known and the level set $\{x : F(x) \leq F(x_0), Ax \geq b\}$ is compact.*

Assumption 2.3. *The matrix of active constraints has full row rank at any point satisfying the second-order necessary optimality conditions. Let $\hat{x}$ be a feasible point. Say A_A is the matrix of active constraints with nullspace matrix Z_A. If*

$$Z_A^T \nabla F(\hat{x}) = 0 \text{ and } \lambda_{\min}(Z_A^T \nabla^2 F(\hat{x}) Z_A) \geq 0$$

then A_A has full row rank.

2.3 Definition of the algorithm

The general arc search algorithm generates a sequence of iterates $\{x_k\}_{k=0}^{\infty}$ with

$$x_{k+1} = x_k + \Gamma_k(\alpha_k),$$

where $\Gamma_k \in C^2 : \mathbb{R} \to \mathbb{R}^n$ is the search arc and α_k is the step size. Iteration k starts at point x_k and ends at x_{k+1}. We denote a_i^T as the ith row of A and b_i as the ith element of b. A constraint $a_i^T x_k \geq b_i$ is said to be *inactive* if $a_i^T x_k > b_i$, *active* if $a_i^T x_k = b_i$, and *violated* if $a_i^T x_k < b_i$. At the start of iteration k the algorithm has access to the following objects:

x_k	current variable or iterate
F_k, g_k, H_k	values for the objective function, gradient, and Hessian
A_k	matrix of active constraints
Z_k	nullspace matrix associated with A_k, i.e. $A_k Z_k = 0$
$\mathcal{W}_k \subseteq \{1, \ldots, m\}$	index set of active constraints, $i \in \mathcal{W}_k$ if and only if $a_i^T x_k = b_i$
π_k	vector of multiplier estimates associated with the active set
$\overline{\Gamma}_k$	hypothetical arc constructed to remain on constraints active at x_k, $A_k \overline{\Gamma}_k(\alpha) = 0$ for all $\alpha \geq 0$

The algorithm will inspect π_k and other data to determine if a constraint can be deleted. After this process, the following objects are defined:

$\bar{A}_k$	matrix of constraints that will remain active during iteration k
$\bar{Z}_k$	nullspace matrix associated with $\bar{A}_k$, i.e. $\bar{A}_k \bar{Z}_k = 0$
$\overline{\mathcal{W}}_k$	index set of constraints that will remain active during iteration k
Γ_k	search arc for iteration k, $A_k \Gamma_k(\alpha) \geq 0$ and $\bar{A}_k \Gamma_k(\alpha) = 0$ for $\alpha \geq 0$

If no constraints are deleted then $\bar{A}_k = A_k$, $\bar{Z}_k = Z_k$, $\overline{\mathcal{W}}_k = \mathcal{W}_k$, and $\Gamma_k = \overline{\Gamma}_k$.

An arc search will then be made on the arc $x_k + \Gamma_k(\alpha)$ to select the step size α_k. The next iterate is then $x_{k+1} = x_k + \Gamma_k(\alpha_k)$. If the step is limited by a constraint, then A_{k+1}, Z_{k+1}, $\mathcal{W}_{k+1}$, and π_{k+1} are updated to account for the change.

Set notation will be used to compare index sets at the start of different iterations. The intersection $\mathcal{W}_{k_1} \cap \mathcal{W}_{k_2} = \{i : i \in \mathcal{W}_{k_1}, i \in \mathcal{W}_{k_2}\}$ contains indices that are in both $\mathcal{W}_{k_1}$ and $\mathcal{W}_{k_2}$. The set difference notation $\mathcal{W}_{k_1} \backslash \mathcal{W}_{k_2} = \{i : i \in \mathcal{W}_{k_1}, i \notin \mathcal{W}_{k_2}\}$ indicates indices that are in $\mathcal{W}_{k_1}$ but not in $\mathcal{W}_{k_2}$.

2.3.1 Properties of Γ_k

The search arc $\Gamma_k : \mathbb{R} \mapsto \mathbb{R}^n$ is twice continuously differentiable with $\Gamma_k(0) = 0$. The arc must provide initial descent, $g_k^T \Gamma_k'(0) \leq 0$, and remain on constraints in $\overline{\mathcal{W}}_k$, i.e. $\bar{A}_k \Gamma_k(\alpha) = 0$ for $\alpha \geq 0$. If step size α is bounded, the arc must be bounded. Formally,

$$\alpha \leq \tau_1 \implies \exists \text{ finite } \tau_2 \text{ such that } \|\Gamma_k'(\alpha)\| \leq \tau_2 \ \forall \ \alpha \in [0, \tau_1]. \tag{2.1}$$

If α is unbounded it is acceptable for the arc and its derivative to diverge. On any subsequence of iterates I, the arc must satisfies the following properties:

First-order convergence

$$\lim_{k\in I} g_k^T\Gamma'_k(0) = 0 \implies \lim_{k\in I} Z_k^T g_k = 0 \tag{2.2}$$

Second-order convergence

$$\liminf_{k\in I} \Gamma'_k(0)^T H_k\Gamma'_k(0) + g_k^T\Gamma''_k(0) \geq 0 \implies \liminf_{k\in I} \lambda_{\min}(Z_k^T H_k Z_k) \geq 0 \tag{2.3}$$

Arc convergence

$$\begin{aligned} \lim_{k\in I} g_k^T\Gamma'_k(0) = 0 \text{ and } \liminf_{k\in I} \Gamma'_k(0)^T H_k\Gamma'_k(0) + g_k^T\Gamma''_k(0) \geq 0 \\ \implies \lim_{k\in I} \Gamma'_k(0) = 0 \end{aligned} \tag{2.4}$$

In (2.3), $\lambda_{\min}(Z_k^T H_k Z_k)$ denotes the minimal eigenvalue of $Z_k^T H_k Z_k$.

2.3.2 Properties of π_k

At each iteration a vector of Lagrange multiplier estimates π_k is computed and must satisfy

$$\lim_{k\in I} \|Z_k^T g_k\| = 0 \implies \lim_{k\in I} \|g_k - A_k^T\pi_k\| = 0. \tag{2.5}$$

The minimum multiplier is denoted $\pi_{\min,k} = \min_i(\pi_k)_i$.

2.3.3 Properties of Γ_k related to constraints

The hypothetical arc $\overline{\Gamma}_k$ is constructed to remain on the set of constraints active at the beginning of iteration k:

$$A_k\overline{\Gamma}_k(\alpha) = 0 \text{ and } A_k\overline{\Gamma}'_k(\alpha) = 0 \text{ for all } \alpha \geq 0.$$

The true search arc Γ_k satisfies the following rules and properties:

Deletion Constrained are only considered for deletion if $\pi_{\min,k} < 0$. If no constraint is to be deleted or a constraint was added in the last iteration, then set $\Gamma_k(\alpha) = \overline{\Gamma}_k(\alpha)$. If the most recent change to the working set was a constraint addition, then only one constraint may be deleted. If the most recent change to the working set was a constraint deletion, then more than one constraint may be deleted. If constraints are to be deleted,

$$A_k\Gamma'_k(0) > 0 \text{ and } A_k\Gamma'_k(0) \neq 0.$$

Descent If constraints are to be deleted the arc must provide initial descent such that

$$g_k^T \Gamma'_k(0) \le g_k^T \overline{\Gamma}'_k(0) \le 0.$$

Convergence It is also required that

$$\lim_{k\in I} g_k^T(\Gamma'_k(0) - \overline{\Gamma}'_k(0)) = 0 \implies \begin{cases} \liminf_{k\in I} \pi_{\min,k} \ge 0 \\ \lim_{k\in I} \|\Gamma_k(\alpha) - \overline{\Gamma}_k(\alpha)\| = 0, \end{cases} \tag{2.6}$$

$$a_i^T \Gamma'_k(0) > 0 \implies (\pi_k)_i \le \nu \pi_{\min,k} \text{ for } k \in I \text{ and } i \in \mathcal{W}_k \backslash \overline{\mathcal{W}}_k, \tag{2.7}$$

where I is any subsequence such that $\overline{\mathcal{W}}_k \subseteq W_k$ for all $k \in I$ and ν is a tolerance in the interval $(0,1]$.

2.3.4 Properties of α_k

At each iteration the univariate search function is defined as

$$\phi_k(\alpha) = F(x_k + \Gamma_k(\alpha)).$$

The first and second derivatives are

$$\begin{aligned} \phi'_k(\alpha) &= \nabla F(x_k + \Gamma_k(\alpha))^T \Gamma'_k(\alpha) \\ \phi''_k(\alpha) &= \Gamma'_k(\alpha)^T \nabla^2 F(x_k + \Gamma_k(\alpha)) \Gamma'_k(\alpha) + \nabla F(x_k + \Gamma_k(\alpha))^T \Gamma''_k(\alpha). \end{aligned}$$

When evaluated at $\alpha = 0$ these reduce to

$$\begin{aligned} \phi'_k(0) &= g_k^T \Gamma'_k(0) \\ \phi''_k(0) &= \Gamma'_k(0)^T H_k \Gamma'_k(0) + g_k^T \Gamma''_k(0). \end{aligned}$$

The step size α_k is computed to satisfy certain conditions that enforce convergence of the algorithm:

Boundedness An upper bound on the step size is computed with

$$\bar{\alpha}_k = \max\{\alpha : A(x_k + \Gamma_k(\alpha)) \ge b \text{ and } \alpha \le \alpha_{\max}\}$$

where $\alpha_{\max}$ is a fixed upper limit. If $\bar{\alpha}_k = 0$, then $\alpha_k = 0$.

Descent condition The selected step size must provide a sufficient reduction in the objective function according to

$$\phi_k(\alpha_k) \le \phi_k(0) + \mu\left(\phi'_k(0)\alpha_k + \tfrac{1}{2}\min\{\phi''_k(0), 0\}\alpha_k^2\right) \tag{2.8}$$

with $0 < \mu \le \frac{1}{2}$.

Curvature condition The selected step size must provide a sufficient reduction in curvature unless it encounters a constraint. Thus, α_k must satisfy at least one of

$$|\phi_k'(\alpha_k)| \le \eta|\phi_k'(0) + \min\{\phi_k''(0), 0\}\alpha_k| \tag{2.9}$$

$$\text{or} \qquad \phi_k'(\alpha_k) < \eta(\phi_k'(0) + \min\{\phi_k''(0), 0\}\alpha_k) \text{ and } \alpha_k = \bar{\alpha}_k \tag{2.10}$$

with $\mu \le \eta < 1$. In the case where α_k is limited by $\bar{\alpha}_k$, (2.10) indicates that the derivative of the search function must be negative. If, in the other case, $\phi_k'(\alpha_k) > \eta|\phi_k'(0) + \min\{\phi_k''(0), 0\}\alpha_k|$ then a robust search routine should reduce α to find a step size satisfying (2.9).

The set of points that satisfy both (2.8) and (2.9) is denoted Φ_k. The step size is called *restricted* if $\alpha_k = \bar{\alpha}_k$ and $\alpha_k < \alpha_{\max}$, which indicates that a constraint has been encountered. Otherwise, the step is called *unrestricted* and satisfies (2.9) or (2.10) with $\bar{\alpha}_k = \alpha_{\max}$. Note that if $\alpha_k \notin \Phi_k$, it must satisfy (2.8) and (2.10).

Moré and Sorensen [48, Lemma 5.2] provide a proof for the existence of a step size satisfying (2.8) and (2.9). For completeness, it is reproduced here.

Lemma 2.1. *Let $\phi : \mathbb{R} \to \mathbb{R}$ be twice continuously differentiable in an open interval Ω that contains the origin, and suppose that $\{\alpha \in \Omega : \phi(\alpha) \le \phi(0)\}$ is compact. Let $\mu \in (0,1)$ and $\eta \in [\mu, 1)$. If $\phi'(0) < 0$, or if $\phi'(0) \le 0$ and $\phi''(0) < 0$, then there is an $\alpha > 0$ in Ω such that*

$$\phi(\alpha) \le \phi(0) + \mu\left(\phi'(0)\alpha + \tfrac{1}{2}\min\{\phi''(0), 0\}\alpha^2\right) \tag{2.11}$$

and

$$|\phi'(\alpha)| \le \eta|\phi'(0) + \min\{\phi''(0), 0\}\alpha|. \tag{2.12}$$

Proof. Let

$$\beta = \sup\{\alpha \in I : \phi(\alpha) \le \phi(0)\}.$$

Then $\beta > 0$ since either $\phi'(0) < 0$, or $\phi'(0) \le 0$ and $\phi''(0) < 0$. Moreover, the compactness assumption and the continuity of ϕ imply that β is finite and that $\phi(0) = \phi(\beta)$. Thus

$$\phi(\beta) \ge \phi(0) + \mu\left(\phi'(0)\beta + \tfrac{1}{2}\min\{\phi''(0), 0\}\beta^2\right).$$

Define $\psi : I \to \mathbb{R}$ by

$$\psi(\alpha) = \phi(\alpha) - \phi(0) - \eta\left(\phi'(0)\alpha + \tfrac{1}{2}\min\{\phi''(0), 0\}\alpha^2\right).$$

Since $\mu \le \eta$ we have $\psi(\beta) \ge 0$. Note also that $\psi(0) = 0$ and either $\psi'(0) < 0$, or $\psi'(0) \le 0$ and $\psi''(0) < 0$. This, together with the continuity of ψ, implies the existence of $\beta_1 \in (0, \beta]$ such that $\psi(\beta_1) = 0$, and $\psi(\alpha) < 0$ for all $\alpha \in (0, \beta_1)$. Now Rolle's theorem shows that there is an $\alpha \in (0, \beta_1)$ with $\psi'(\alpha) = 0$, and thus (2.12) follows. Moreover, $\psi(\alpha) < 0$ and $\mu \le \eta$ imply (2.11). $\square$

2.3.5 Properties of A_k

The matrix of active constraints A_k is required to have full row rank for all k. Recall that $\overline{\mathcal{W}}_k$ is the index set of constraints that will remain active during iteration k. Let $\mathcal{P}_k^a$ be the index set of constraints that become active at the end of iteration k,

$$\mathcal{P}_k^a = \{i \notin \overline{\mathcal{W}}_k : a_i^T x_{k+1} = b_i\}.$$

The working set at the start of iteration $k+1$ is $\mathcal{W}_{k+1} = \overline{\mathcal{W}}_k \cup \mathcal{W}_k^a$, where $\mathcal{W}_k^a \subseteq \mathcal{P}_k^a$ and A_{k+1} are required to satisfy

$$\mathcal{P}_k^a \neq \emptyset \implies \mathcal{W}_k^a \neq \emptyset \quad \text{and} \tag{2.13}$$

$$A_{k+1} \quad \text{has full row rank.} \tag{2.14}$$

Thus if constraints are encountered then at least one must be added to the working set. Note that a step is restricted if and only if $\mathcal{W}_{k+1} \backslash \overline{\mathcal{W}}_k \neq \emptyset$.

The nullspace matrix Z_k is required to have a bounded condition number for all k.

2.4 Convergence results

The convergence results for this arc search method generalize the work by Forsgren and Murray on a line search method [28]. Lemmas 2.2, 2.3, and 2.4 establish convergence on subsequences of iterations with unrestricted steps and are based on Lemmas 4.1, 4.2, and 4.3 in [28]. The original result for unconstrained optimization with a curvilinear search comes from Moré and Sorensen [48].

Lemma 2.2. *Given assumptions 2.1–2.3, assume that a sequence* $\{x_k\}_{k=0}^{\infty}$ *is generated as outlined in section 2.3. Then*

(i) $\lim_{k\to\infty} \alpha_k \phi_k'(0) = 0$,

(ii) $\lim_{k\to\infty} \alpha_k^2 \min\{\phi_k''(0), 0\} = 0$,

(iii) $\lim_{k\to\infty} \|x_{k+1} - x_k\| = 0$.

Proof. Rearrangement of (2.8) gives

$$\phi_k(0) - \phi_k(\alpha_k) \geq -\mu\left(\phi_k'(0)\alpha_k + \tfrac{1}{2}\min\{\phi_k''(0), 0\}\alpha_k^2\right).$$

Since $\mu > 0$, $\phi_k'(0) \leq 0$, and the objective function is bounded from below on the feasible region, (i) and (ii) follow.

To show (iii), we write $x_{k+1} - x_k = \Gamma_k(\alpha_k)$ and show that $\lim_{k\to\infty} \|\Gamma_k(\alpha_k)\| = 0$. $\Gamma_k(\alpha)$ is continuous with $\Gamma_k(0) = 0$. Also α_k and $\|\Gamma_k'(0)\|$ are bounded. Therefore, if $\lim_{k\to\infty} \|\Gamma_k(\alpha_k)\| \neq 0$, there must exist a subsequence I with $\epsilon_1 > 0$ and $\epsilon_2 > 0$ such that $\alpha_k \geq \epsilon_1$ and $\|\Gamma_k'(0)\| \geq \epsilon_2$ for all $k \in I$. From the existence of ϵ_1, (i) implies $\lim_{k\in I} \phi_k'(0) = 0$ and (ii) implies $\lim_{k\in I} \phi_k''(0) \geq 0$.

Since $\phi_k'(0) = g_k^T\Gamma_k'(0)$ and $\phi_k''(0) = \Gamma_k'(0)^T H_k \Gamma_k'(0) + g_k^T\Gamma_k''(0)$, the termination property of Γ_k (2.4) implies that $\lim_{k\in I} \|\Gamma_k'(0)\| = 0$. This contradicts the existence of ϵ_2, thus establishing (iii). □

Lemma 2.3. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. If, at iteration k, an unrestricted step is taken, then either $\alpha_k = \alpha_{\max}$ or there exists a $\theta_k \in (0, \alpha_k)$ such that*

$$\alpha_k \left(\phi_k''(\theta_k) + \eta \max\{-\phi_k''(0), 0\}\right) \geq -(1-\eta)\phi_k'(0). \tag{2.15}$$

Proof. Since $\phi_k'(0) \leq 0$, it follows from (2.9) if α_k is unrestricted and $\alpha_k < \alpha_{\max}$, it satisfies

$$-\phi_k'(\alpha_k) \leq -\eta\phi_k'(0) + \eta \max\{-\phi_k''(0), 0\}\alpha_k. \tag{2.16}$$

Further, since ϕ_k' is a continuously differentiable univariate function, the mean-value theorem ensures the existence of a $\theta_k \in (0, \alpha_k)$ such that

$$\phi_k'(\alpha_k) = \phi_k'(0) + \alpha_k\phi_k''(\theta_k). \tag{2.17}$$

A combination of (2.16) and (2.17) gives (2.15). □

Lemma 2.4. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. Let I denote a subsequence of iterations where unrestricted steps are taken; then*

(i) $\lim_{k\in I} \phi_k'(0) = 0$,

(ii) $\lim_{k\in I} \phi_k''(0) \geq 0$,

(iii) $\lim_{k\in I} Z_k^T g_k = 0$ and $\liminf_{k\in I} \lambda_{\min}(Z_k^T H_k Z_k) \geq 0$.

Proof. To show (i), assume by contradiction there is a subsequence $I' \subseteq I$ such that $\phi_k'(0) \leq \epsilon_1 < 0$ for $k \in I'$. Lemma 4.2 in conjunction with assumptions A1 and A2 then implies that $\limsup_{k\in I'} \alpha_k \neq 0$, contradicting Lemma 4.1. Hence, the assumed existence of I' is false, and we conclude that (i) holds.

Similarly, to show (ii), assume by contradiction that there is a subsequence $I'' \subseteq I$ such that $\phi_k''(0) \leq \epsilon_2 < 0$ for $k \in I''$. Since $\alpha_k > 0$ and $\phi_k'(0) \leq 0$, Lemma 2.3 implies that for $k \in I''$ there exists $\theta_k \in (0, \alpha_k)$ such that

$$\phi_k''(\theta_k) - \eta\phi_k''(0) \geq 0. \tag{2.18}$$

Lemma 2.2 gives $\lim_{k\in I''} \alpha_k = 0$, and thus (2.18) cannot hold for k sufficiently large. Consequently, the assumed existence of I'' is false, and (ii) holds.

Finally, we show that (i) and (ii) imply (iii). (i) and the sufficient descent property of the arc (2.2) imply $\lim_{k\in I} Z_k^T g_k = 0$. (i) and the sufficient negative curvature property of the arc (2.3) imply $\liminf_{k\in I} \lambda_{\min}(Z_k^T H_k Z_k) \geq 0$. □

We now consider linear constraints. The following lemma states that multipliers π_k and π_{k+1} must converge on a subsequence of iterations with unrestricted steps and the same set of active constraints.

Lemma 2.5. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. Let I denote a subsequence of iterations. If $\alpha_k \in \Phi_k$ (α_k is unrestricted) and $\mathcal{W}_k = \mathcal{W}_{k+1} = \mathcal{W}^I$ (working set does not change) for $k \in I$, then $\lim_{k\in I} \|\pi_{k+1} - \pi_k\| = 0$.*

Proof. Denote A_I as the matrix of active constraints for $k \in I$. Assume for contradiction there is a subsequence $I' \subseteq I$ such that $\|\pi_{k+1} - \pi_k\| \geq \epsilon$ for $k \in I'$. Part (iii) of Lemma 2.4 and part (iii) of Lemma 2.2 imply that $\lim_{k\in I'} Z_k^T g_k = 0$ and $\lim_{k\in I'} Z_{k+1}^T g_{k+1} = 0$. (2.5) implies that $\lim_{k\in I'} \|g_k - A_I^T \pi_k\| = 0$ and $\lim_{k\in I'} \|g_{k+1} - A_I^T \pi_{k+1}\| = 0$. Combining the previous two limits we see

$$\lim_{k\in I'} \|(g_{k+1} - g_k) - A_I^T(\pi_{k+1} - \pi_k)\| = 0. \tag{2.19}$$

However, $\|\pi_{k+1} - \pi_k\| \geq \epsilon$ for $k \in I'$ and (2.19) imply the existence of a K such that $\pi_{k+1} - \pi_k \in \operatorname{null}(A_I)$ and $\operatorname{null}(A_I) \neq \emptyset$ for all $k \geq K$ and $k \in I'$. This contradicts assumption A3 that matrix A_I has full rank. Thus, the assumed existence of subsequence I' is false, and we must have $\lim_{k\in I} \|\pi_{k+1} - \pi_k\| = 0$. □

The following lemma states that constraints will eventually be encountered on a subsequence of iterations where constraints are deleted, the minimal multiplier is negative and bounded away from zero, and for which constraints are not deleted in the previous iteration. This result is derived from Lemma 4.4 in [28]. In this case, arcs may move from and encounter the same constraint in a single iteration. Therefore the consequence of the lemma presented here allows $\mathcal{W}_{k+1} = \mathcal{W}_k$ for $k \in I$ and $k \geq K$, which is not possible in the line search case.

Lemma 2.6. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. If there is a subsequence I and an $\epsilon > 0$ such that $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$, $\Gamma_k \neq \overline{\Gamma}_k$, and $\pi_{\min,k} < -\epsilon$ for $k \in I$, then there is an integer K such that $\mathcal{W}_{k+1} \backslash \overline{\mathcal{W}}_k \neq \emptyset$ for $k \in I$ and $k \geq K$.*

Proof. In parts for clarity:

1. Suppose there is a subsequence I and an $\epsilon > 0$ such that $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$, $\Gamma_k \neq \overline{\Gamma}_k$, and $\pi_{\min,k} < -\epsilon$ for $k \in I$. No constraints are deleted in iteration $k-1$ while at least one constraint is deleted in iteration k.

2. Assume there is a subsequence $I' \subseteq I$ such that an unrestricted step is taken for $k \in I'$.

3. Lemma 2.4 implies that $\lim_{k\in I'} \phi_k'(0) = 0$.

4. On the other hand, (2.6) ensures the existence of a subsequence $I'' \subseteq I'$ and a positive constant ϵ_2 such that $g_k^T(\Gamma_k'(0) - \overline{\Gamma}_k'(0)) \leq -\epsilon_2$ for all $k \in I''$.

5. However, $g_k^T \overline{\Gamma}_k'(0) \leq 0$ implies that $\phi_k'(0) \leq -\epsilon_2$ for all $k \in I''$, which is a contradiction.

6. Hence, the assumed existence of subsequence I' is false, and there must exist a K such that for all $k \in I$ and $k \geq K$ a restricted step is taken. ($\mathcal{W}_{k+1} \backslash \overline{\mathcal{W}}_k \neq \emptyset$ for all $k \in I$ and $k \geq K$.)

□

The following lemma shows that if there exists a subsequence of iterations where the same constraint is both deleted and added, then the multiplier corresponding to the constraint converges to a positive value.

Lemma 2.7. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. If there exists a subsequence I such that for $k \in I$*

- $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$ *(no constraint is deleted in iteration $k-1$)*
- $\pi_{\min,k} \leq -\epsilon_1$ *and the constraint corresponding to $(\pi_k)_i$ is deleted ($\Gamma_k \neq \overline{\Gamma}_k$)*
- *in iteration k the arc moves off, but is restricted by, the constraint corresponding to $(\pi_k)_i$, with $\alpha_k \notin \Phi_k$*
- *in iteration $k+1$, no constraint is deleted ($\Gamma_{k+1} = \overline{\Gamma}_{k+1}$) and the step is unrestricted ($\alpha_{k+1} \in \Phi_{k+1}$)*

then $\liminf_{k \in I} (\pi_{k+1})_i > 0$.

Proof. In parts for clarity:

1. Assume for contradiction the existence of a subsequence $I' \subseteq I$ such that $(\pi_{k+1})_i \leq 0$.
2. At the beginning of iteration k, constraint i was deleted. In order for the same constraint to restrict the step size we must have $a_i^T \Gamma_k'(\alpha_k) \leq 0$.
3. (2.6) implies the existence of a subsequence $I'' \subseteq I'$ and a positive constant ϵ_2 such that $g_k^T \Gamma_k'(0) \leq -\epsilon_2$ for $k \in I''$.
4. Constraint i limits the step size and enforces $\alpha_k \notin \Phi_k$. With (2.10) we have $\phi_k'(\alpha_k) < \eta \phi_k'(0)$ or $g_{k+1}^T \Gamma_k'(\alpha_k) < \eta g_k^T \Gamma_k'(0) \leq -\eta\epsilon_2$ for $k \in I''$.
5. The gradient at the start of iteration $k+1$ can be represented as $g_{k+1} = A_{k+1}^T \pi_{k+1} + r_{k+1}$ where r_{k+1} is the residual.
6. We have

$$g_{k+1}^T \Gamma_k'(\alpha_k) = \pi_{k+1}^T A_{k+1} \Gamma_k'(\alpha_k) + r_{k+1}^T \Gamma_k'(\alpha_k) \tag{2.20}$$
$$= (\pi_{k+1})_i a_i^T \Gamma_k'(\alpha_k) + r_{k+1}^T \Gamma_k'(\alpha_k) < -\eta\epsilon_2 \tag{2.21}$$

   for $k \in I''$.

7. Combining $(\pi_{k+1})_i \leq 0$ and $a_i^T \Gamma_k'(\alpha_k) \leq 0$ we see that $r_{k+1}^T \Gamma_k'(\alpha_k) < -\eta\epsilon_2$ for $k \in I''$.

8. Apply the Cauchy-Schwartz inequality and we see

$$\|r_{k+1}\|\|\Gamma'_k(\alpha_k)\| > \eta\epsilon_2 \tag{2.22}$$

or

$$\|\Gamma'_k(\alpha_k)\| > \frac{\eta\epsilon_2}{\|r_{k+1}\|} \tag{2.23}$$

for $k \in I''$.

9. By assumption, iteration $k+1$ is an unrestricted step. Thus $\lim_{k\in I} \|Z_{k+1}^T g_{k+1}\| = 0$, which implies $\lim_{k\in I} \|g_{k+1} - A_{k+1}^T \pi_{k+1}\| = 0$. We have

$$\lim_{k\in I''} \|r_{k+1}\| = 0 \implies \lim_{k\in I''} \|\Gamma'_k(\alpha_k)\| = \infty \implies \lim_{k\in I''} \alpha_k = \infty,$$

where the first step comes from (2.23) and the second step comes from (2.1). Divergence of α_k contradicts assumption A1. Thus the assumed subsequence I' does not exist and we must have

$$\liminf_{k\in I} (\pi_{k+1})_i > 0.$$

□

The following lemma establishes convergence of the minimal multiplier to a nonnegative value on a subsequence of iterations where constraints are deleted, but not deleted in the previous iteration. It is derived from Lemma 4.5 from [28].

Lemma 2.8. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. If there is a subsequence I such that $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$ and $\Gamma_k \neq \overline{\Gamma}_k$ for $k \in I$, then* $\liminf_{k\in I} \pi_{\min,k} \geq 0$.

Proof. In parts for clarity:

1. Assume that there exists a subsequence I and an $\epsilon > 0$ such that $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$, $\Gamma_k \neq \overline{\Gamma}_k$, and $\pi_{\min,k} < -\epsilon$ for $k \in I$.

2. For each $k \in I$, let l_k denote the following iteration with least index such that $\mathcal{W}_{l_k} = \overline{\mathcal{W}}_{l_k-1} = \mathcal{W}_{l_k-1}$; i.e., an unrestricted step is taken at iteration $l_k - 1$ and $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$ (no constraint is removed in iteration $l_k - 1$).

3. Lemma 2.6 implies that there is an integer K such that $\mathcal{W}_{k+1}\backslash\overline{\mathcal{W}}_k \neq \emptyset$ for all $k \in I$ and $k \geq K$.

4. The properties of Γ_k from section 2.3.3 imply that $\Gamma_{k+1} = \overline{\Gamma}_{k+1}$ for $k \in I$, $k \geq K$.

5. Consequently, for $k \geq K$, $l_k - 1$ is the iteration with least index following k where no constraint is added in the arc search.

6. Since there can be at most $\min\{m, n\}$ consecutive iterations where a constraint is added, it follows from (iii) of Lemma 2.2 that $\lim_{k\in I} \|x_k - x_{l_k}\| = 0$.

7. Consequently, there must exist a point $\bar{x}$ that is a common limit point for $\{x_k\}_{k\in I}$ and $\{x_{l_k}\}_{k\in I}$.

8. Thus, there exists a subsequence $I' \subseteq I$ such that $\lim_{k\in I'} x_k = \bar{x}$ and $\lim_{k\in I'} x_{l_k} = \bar{x}$.

9. Thus, there must exist a subsequence $I'' \subseteq I'$ such that $\mathcal{W}_k$ is identical for every $k \in I''$ and $\mathcal{W}_{l_k}$ is identical for every $l_k \in J$, where J denotes the subsequence $\{l_k\}_{k\in I''}$. Define $\mathcal{W}^I \equiv \mathcal{W}_k$ for any $k \in I''$ and $\mathcal{W}^J \equiv \mathcal{W}_{l_k}$ for any $l_k \in J$.

10. Since all constraints corresponding to $\mathcal{W}^I$ are active at $\bar{x}$ and an infinite number of unrestricted steps are taken where the working set is constant, it follows from assumptions A1 and A2 in conjunction with (iii) of Lemma 2.2 and (iii) of Lemma 2.4 that $\lim_{k\in I''} Z_I^T g_k = 0$ and $\liminf_{k\in I''} \lambda_{\min}(Z_I^T H_k Z_I) \geq 0$, where Z_I denotes a matrix whose columns form an orthonormal basis for the null space of A_I, the constraint matrix associated with $\mathcal{W}^I$.

11. Consequently, (2.5) and the full row rank of A_I^T imply that $\lim_{k\in I''} \pi_k = \pi^I$, where π^I satisfies
$$\nabla f(\bar{x}) = A_I^T \pi^I = \sum_{i\in\mathcal{W}^I} a_i \pi_i^I. \tag{2.24}$$

12. By a similar reasoning and notation for Z_J and A_J we have $\lim_{k\in I''} Z_J^T g_{l_k} = 0$, $\liminf_{k\in I''} \lambda_{\min}(Z_J^T H_{l_k} Z_J) \geq 0$, and $\lim_{k\in I''} \pi_{l_k} = \pi^J$, where π^J satisfies
$$\nabla f(\bar{x}) = A_J^T \pi^J = \sum_{i\in\mathcal{W}^J} a_i \pi_i^J. \tag{2.25}$$

13. Combining (2.24) and (2.25) we obtain
$$\sum_{i\in\mathcal{W}^I\setminus\mathcal{W}^J} a_i\pi_i^I + \sum_{i\in\mathcal{W}^I\cap\mathcal{W}^J} a_i(\pi_i^I - \pi_i^J) + \sum_{i\in\mathcal{W}^J\setminus\mathcal{W}^I} a_i\pi_i^J = 0. \tag{2.26}$$

14. By assumption A3, the vectors a_i, $i \in \mathcal{W}^I \cup \mathcal{W}^J$ are linearly independent. Hence, it follows from (2.26) that
$$\pi_i^I = 0 \qquad \text{for } i \in \mathcal{W}^I\setminus\mathcal{W}^J \tag{2.27}$$
$$\pi_i^I = \pi_i^J \qquad \text{for } i \in \mathcal{W}^I\cap\mathcal{W}^J \tag{2.28}$$
$$\pi_i^J = 0 \qquad \text{for } i \in \mathcal{W}^J\setminus\mathcal{W}^I. \tag{2.29}$$

15. Since no constraints have been deleted between iterations k and l_k for $k \in I''$, any constraint whose index is in the set $\mathcal{W}^I\setminus\mathcal{W}^J$ must have been deleted in an iteration $k \in I''$. Since $I'' \subseteq I$, it follows that $\pi_{\min,k} \leq -\epsilon$ for $k \in I''$. From the rule for moving off a constraint, (2.7), we can

deduce that $(\pi_k)_i \leq -\nu\epsilon$ for $k \in I''$ and $i \in \mathcal{W}^I \backslash \mathcal{W}^J$, where $\nu \in (0,1)$. Since $\lim_{k\in I''} \pi_k = \pi^I$, we conclude that $\pi_i^I \leq -\nu\epsilon$ for $i \in \mathcal{W}^I \backslash \mathcal{W}^J$. Hence, (2.27) implies that $\mathcal{W}^I \backslash \mathcal{W}^J = \emptyset$.

16. If constraint i is deleted in iteration $k \in I''$, then it must be added before iteration $l_k - 1$ because $\mathcal{W}^I \backslash \mathcal{W}^J = \emptyset$. Again, we can deduce that $(\pi_k)_i \leq -\nu\epsilon$ and $\pi_i^I \leq -\nu\epsilon$ for $i \in \mathcal{W}^I \cap \mathcal{W}^J$. If constraint i is also added in iteration $k \in I''$ ($i \in \mathcal{W}_{k+1}$) then Lemma 2.7 implies that $\liminf_{k\in I''}(\pi_{k+1})_i > 0$. Lemma 2.5 and (2.28) imply that $k + 1 < l_k - 1$. Consequently, it must hold that $|\mathcal{W}^J| \geq |\mathcal{W}^I| + 1$ and, by (2.29), π^J has at least one zero element.

17. We can conclude from (2.28) that $\pi_{\min,l_k} < -0.5\epsilon$ for $k \in I''$ and k sufficiently large. The rules for computing Γ_k, (2.6), ensure that there is a subsequence $I''' \subseteq I''$ such that $\Gamma'_{l_k}(0) \neq 0$ for all $k \in I'''$. From the definition of l_k, it holds that $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$ for all $k \in I'''$. Therefore, if $J' = \{l_k : k \in I'''\}$, we may replace I by J' and repeat the argument. Since $|\mathcal{W}^J| \geq |\mathcal{W}^I| + 1$ and $|\mathcal{W}_k| \leq \min\{m,n\}$ for any k, after having repeated the argument at most $\min\{m,n\}$ times we have a contradiction to assumption A3, implying that the assumed existence of a subsequence I such that $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$ and $\Gamma_k \neq \overline{\Gamma}_k$ and $\pi_{\min,k} < -\epsilon$ for $k \in I$ is false.

□

The following theorem gives the main convergence result. It is derived from Theorem 4.6 in [28].

Theorem 2.1. *Given assumptions 2.1–2.3, assume that a sequence $\{x_k\}_{k=0}^{\infty}$ is generated as outlined in section 2.3. Then, any limit point x^* satisfies the second-order necessary optimality conditions; i.e., if the constraint matrix associated with the active constraints at x^* is denoted by A_A, there is a vector π_A such that*

$$\nabla f(x^*) = A_A^T \pi_A, \quad \pi_A \geq 0,$$

and it holds that

$$\lambda_{\min}(Z_A^T \nabla^2 f(x^*) Z_A) \geq 0,$$

where Z_A denotes a matrix whose columns form a basis for the null space of A_A.

If in addition $\lambda_{\min}(Z_A^T \nabla^2 f(x^*) Z_A) > 0$ and $\pi_A > 0$, then $\lim_{k\to\infty} x_k = x^*$. Further, for k sufficiently large, it follows that if $\Gamma_k(\alpha_k) = -Z_A(Z_A^T H_k Z_A)^{-1} Z_A^T g_k$ then Γ_k and α_k satisfy (2.8) and (2.9). Moreover, for this choice of Γ_k and α_k, the rate of convergence is at least q-quadratic, provided the second-derivative matrix is Lipschitz continuous in a neighborhood of x^*.

Proof. In parts for clarity:

1. Let x^* denote a limit point of a generated sequence of iterates.

2. By assumption A2, there is a subsequence I such that $\lim_{k\in I} x_k = x^*$.

3. **Claim:** this implies existence of subsequence I' such that $\lim_{k\in I'} x_k = x^*$, $\Gamma_{k-1} = \overline{\Gamma}_{k-1}$ and $A_{k-1} = A_k = A^*$ for each $k \in I$, where A^* denotes a matrix that is identical for each $k \in I'$ and defines the active set at x^*. (There exists a subsequence where no constraint is added or deleted and the working set is the same.)

4. For $k \in I$, an iterate l_k is defined as follows:
   (a) If $\Gamma_k \neq \overline{\Gamma}_k$, let l_k be the iteration with largest index that does not exceed k for which $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$. Since no constraints are deleted immediately upon adding constraints, we obtain $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$, $\Gamma_{l_k} \neq \overline{\Gamma}_{l_k}$, $\mathcal{W}_{l_k-1} = \mathcal{W}_{l_k}$, and $k - m \leq l_k \leq k$. Here, constraints are deleted in iterations l_k to k. No constraints are added in iterations $l_k - 1$ to $k - 1$.
   (b) If $\Gamma_k = \overline{\Gamma}_k$, let l_k denote the iteration with least index following k such that $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$ and $\mathcal{W}_{l_k-1} = \mathcal{W}_{l_k}$. Since no constraints are deleted immediately upon adding constraints, it follows that $l_k - 1$ is the iteration with least index when no constraint is added. For this case, we obtain $\Gamma_{l_k-1} = \overline{\Gamma}_{l_k-1}$, $\mathcal{W}_{l_k-1} = \mathcal{W}_{l_k}$, and $k + 1 \leq l_k \leq k + m$.
   (c) It follows from (iii) of Lemma 2.2 that $\lim_{k\in I} \|x_k - x_{l_k}\| = 0$, and hence $\lim_{k\in I} x_{l_k} = x^*$. With $\{l_k\}_{k\in I}$ defined this way, since there is only a finite number of different active-set matrices, the required subsequence I' can be obtained as a subsequence of $\{l_k\}_{k\in I}$.

5. Since, for each $k \in I'$, an unrestricted step is taken at iteration $k - 1$, assumptions A1 and A2 in conjunction with property (iii) of Lemma 2.4 give

$$\hat{Z}^T \nabla f(x^*) = 0 \text{ and } \lambda_{\min}(\hat{Z}^T \nabla^2 f(x^*)\hat{Z}) \geq 0, \tag{2.30}$$

   where $\hat{Z}$ denotes an matrix whose columns form a basis for the null space of $\hat{A}$. Since $\lim_{k\in I'} \hat{Z}^T g_k = 0$ and $\hat{A}$ has full row rank, it follows from (2.5) and (2.30) that

$$\nabla f(x^*) = \hat{A}^T \hat{\pi} \text{ for } \hat{\pi} = \lim_{k\in I'} \pi_k. \tag{2.31}$$

6. It remains to show that $\min_i \hat{\pi}_i \geq 0$. Assume that there is a subsequence $I'' \subseteq I'$ and an $\epsilon > 0$ such that $\pi_{\min,k} < -\epsilon$ for $k \in I''$. Lemma 2.8 shows that there exists a K such that $\Gamma_k = \overline{\Gamma}_k$ for $k \in I''$ and $k \geq K$. But this contradicts (2.6), and since $\hat{\pi} = \lim_{k\in I'} \pi_k$, we conclude that

$$\min_i \hat{\pi}_i \geq 0. \tag{2.32}$$

7. A combination of (2.30), (2.31), (2.32) now ensures that x^* satisfies the second-order necessary optimality conditions. If there are constraints in A_A that are not in $\hat{A}$, the associated multipliers are zero, i.e. π_A equals $\hat{\pi}$ possibly extended by zeros. Also, in this situation, the range space of Z_A is contained in the range space of $\hat{Z}$. Hence, $\lambda_{\min}(\hat{Z}^T \nabla^2 f(x^*)\hat{Z}) \geq 0$ implies $\lambda_{\min}(Z_A^T \nabla^2 f(x^*) Z_A) \geq 0$.

8. To show the second part of the theorem, note that if $\pi_A > 0$, then we must have $\hat{\pi} = \pi_A$, and it follows from (2.31) that there cannot exist a subsequence $\tilde{I}' \subseteq I'$ such that $\pi_{\min,k} < 0$ for $k \in \tilde{I}'$. This implies that there is an iteration $\tilde{K}$ such that $A_k = \hat{A}$ and $\Gamma_k = \overline{\Gamma}_k$. Then the

problem may be written as an equality-constrained problem in the null space of $\hat{A}$, namely

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & \hat{A}x = \hat{b}, \end{array} \tag{2.33}$$

where $\hat{b}$ denotes the corresponding subvector of b.

9. If $\hat{Z}^T\nabla^2F(x^*)\hat{Z}$ is positive definite, then (iii) of Lemma 2.2 and (2.8) ensure that the limit point is unique, i.e., $\lim_{k\to\infty} x_k = x^*$. From the continuity of F, it follows that $\hat{Z}^T H_k \hat{Z}$ is positive definite for k sufficiently large. If Γ_k is constructed such that there exists some α^N with $\Gamma_k(\alpha^N) = -Z_A(Z_A^T H_k Z_A)^{-1} Z_A^T g_k$, then it follows from Bertsekas [4, p. 78] that $\alpha_k = \alpha^N$ eventually satisfies (2.8) and (2.9). This choice of $\Gamma_k(\alpha^N)$ is the Newton step for (2.33). Bertsekas [4, p. 90] also shows that under these conditions $\lim_{k\to\infty} x_k = x^*$ and the rate of convergence is q-quadratic provided $\nabla^2F(x)$ is Lipschitz continuous in a neighborhood of x^*.

$\square$

Chapter 3

Arcs

3.1 Preliminaries

This chapter discusses the application of the convergence theory to different arcs. We start by defining several vectors used throughout the treatment. Line and curvilinear search methods compute descent directions s_k sufficient for first-order convergence. All methods use directions of negative curvature d_k for second-order convergence. Forsgren and Murray define a vector q_k to handle constraint deletion [28, Section 3.4], which may be used in both line and curvilinear search.

Definition 3.1. *Vector s_k is said to be a direction of sufficient descent if $g_k^T s_k \le 0$ and*

$$\lim_{k\in I} g_k^T s_k = 0 \implies \lim_{k\in I} Z_k^T g_k = 0 \text{ and } \lim_{k\in I} s_k = 0 \tag{3.1}$$

for any subsequence I.

Definition 3.2. *Vector d_k is said to be a direction of sufficient negative curvature if $d_k^T H_k d_k \le 0$, $g_k^T d_k \le 0$, and*

$$\lim_{k\in I} d_k^T H_k d_k = 0 \implies \liminf_{k\in I} \lambda_{\min}(Z_k^T H_k Z_k) \ge 0 \text{ and } \lim_{k\in I} d_k = 0 \tag{3.2}$$

for any subsequence I.

Definition 3.3. *Vector q_k is said to be a direction of constraint deletion if $g_k^T q_k \le 0$ and $A_k q_k \ge 0$. Also, q_k must have a bounded norm and satisfy*

$$\lim_{k\in I} g_k^T q_k = 0 \implies \liminf_{k\in I} \pi_{\min,k} \ge 0 \quad \text{and} \quad \lim_{k\in I} q_k = 0, \tag{3.3}$$

$$a_i^T q_k > 0 \implies (\pi_k)_i \le \nu \pi_{\min,k} \quad \text{for} \quad k \in I,\ i \in \mathcal{W}_k \backslash \overline{\mathcal{W}}_k, \tag{3.4}$$

where I is any subsequence and ν is a fixed tolerance in the interval $(0, 1]$.

Here, both s_k and d_k are constructed to remain on the constraints active at the beginning of iteration k, i.e., $A_k s_k = 0$ and $A_k d_k = 0$. To satisfy the requirements of Section 2.3.3, $q_k = 0$ if a constraint was encountered in the previous iteration. If the most recent change to the working set was a constraint addition, then only one constraint may be deleted.

For efficiency, modified gradient flow and NEM arcs are computed on low-dimensional subspaces. The following lemmas are used to establish convergence properties when defining an arc on a subspace.

Lemma 3.1. *Let $Q \in \mathbb{R}^{n\times m}$ with $n \geq m$ and $Q^TQ = I$. If $u \in \operatorname{range}(Q)$ then $QQ^Tu = u$.*

Proof.

$$\begin{aligned} u \in \operatorname{range}(Q) &\implies \exists\, y : Qy = u \implies Q^TQy = Q^Tu \\ &\implies y = Q^Tu \implies Qy = QQ^Tu \implies u = QQ^Tu. \end{aligned}$$

□

Lemma 3.2. *Say $A \in \mathbb{R}^{n\times n}$ is nonsingular and $B \in \mathbb{R}^{n\times m}$ has full rank with $n \geq m$. If $Ax = y$ and $x \in \operatorname{range}(B)$ then there exists a unique $z \in \mathbb{R}^m$ that solves*

$$B^TABz = B^Ty \text{ and} \tag{3.5}$$

$$Bz = x. \tag{3.6}$$

Proof. Together, $x \in \operatorname{range}(B)$ and $\operatorname{rank}(B) = m$ imply that there exists a unique $z \in \mathbb{R}^m$ that solves (3.6). By substitution we see that this z solves (3.5):

$$B^TABz = B^Ty \implies B^TAx = B^Ty \implies B^T(Ax - y) = 0.$$

B^TAB is nonsingular, which implies that z is also a unique solution to (3.5). □

3.2 Line search

Forsgren and Murray proved convergence to second-order critical points with a line search. The arc is simply

$$\Gamma_k(\alpha) = \alpha(s_k + d_k). \tag{3.7}$$

The derivatives of the search function at $\alpha = 0$ are

$$\phi_k'(0) = g_k^T(s_k + d_k) \tag{3.8}$$

$$\phi_k''(0) = (s_k + d_k)^T H_k (s_k + d_k). \tag{3.9}$$

Parts (i) and (ii) of Lemma 2.4 from Chapter 2 state that $\lim_{k\in I} \phi_k'(0) = 0$ and $\liminf_{k\in I} \phi_k''(0) \geq 0$, where I is a subsequence of iterations with unrestricted steps. These combined with (3.8) and (3.9) result in $\lim_{k\in I} g_k^T s_k = 0$ and $\liminf_{k\in I} d_k^T H_k d_k \geq 0$, which establishes second-order convergence.

For problems with linear inequality constraints, Forsgren and Murray [28] construct q_k, a direction of constraint deletion satisfying Definition 3.3. If no constraints are deleted in iteration k then $q_k = 0$. If constraints are deleted, q_k must be a descent direction that moves from at least one constraint. To fit with the requirements of Section 2.3.3, the hypothetical arc is constructed as

$$\overline{\Gamma}_k(\alpha) = \alpha(s_k + d_k), \tag{3.10}$$

while the search arc is

$$\Gamma_k(\alpha) = \alpha(s_k + d_k + q_k). \tag{3.11}$$

We see that $\Gamma_k(\alpha) - \overline{\Gamma}_k(\alpha) = \alpha q_k$ and $\Gamma_k'(\alpha) - \overline{\Gamma}_k'(\alpha) = q_k$. Properties (3.3) and (3.4) of q_k satisfy the general arc search requirements (2.6) and (2.7).

It is a simple matter to compute the point of intersection between a line and linear constraint. Let $p_k = s_k + d_k + q_k$. The intersection between $a_i^T x \geq b_i$ and $x_k + \alpha p_k$ occurs at

$$\alpha = \frac{b_i - a_i^T x_k}{a_i^T p_k}.$$

3.3 Curvilinear search

3.3.1 Moré & Sorensen

Moré and Sorensen [48] define the curvilinear arc

$$\Gamma_k(\alpha) = \alpha^2 s_k + \alpha d_k. \tag{3.12}$$

The derivatives of the search function at $\alpha = 0$ are

$$\phi_k'(0) = g_k^T d_k \tag{3.13}$$
$$\phi_k''(0) = d_k^T H_k d_k + 2 g_k^T s_k. \tag{3.14}$$

Part (ii) of Lemma 2.4 combined with $g_k^T s_k \leq 0$ and (3.14) result in $\lim_{k\in I} g_k^T s_k = 0$ and $\liminf_{k\in I} d_k^T H_k d_k \geq 0$. The same arc can be applied to linearly constrained problems by using a vector of constraint deletion q_k. The hypothetical and true search arcs,

$$\overline{\Gamma}_k(\alpha) = \alpha^2 s_k + \alpha d_k \tag{3.15}$$
$$\Gamma_k(\alpha) = \alpha^2 s_k + \alpha(d_k + q_k), \tag{3.16}$$

satisfy the convergence requirements in the same manner as the line search method presented in Section 3.2. Points of arc-constraint intersection are computed by finding non-negative real roots of

$$\alpha^2 a_i^T s_k + \alpha a_i^T d_k + a_i^T x_k - b_i = 0.$$

3.3.2 Goldfarb

Goldfarb [38] defines the search arc

$$\Gamma_k(\alpha) = \alpha s_k + \alpha^2 d_k. \tag{3.17}$$

The derivatives of the search function at $\alpha = 0$ are

$$\phi'_k(0) = g_k^T s_k \tag{3.18}$$
$$\phi''_k(0) = s_k^T H_k s_k + 2 g_k^T d_k. \tag{3.19}$$

The convergence theory of Chapter 2 cannot be directly applied because the sequences $\{\phi'_k(0)\}_{k\in I}$ and $\{\phi''_k(0)\}_{k\in I}$ do not ensure $\liminf_{k\in I} d_k^T H_k d_k \geq 0$. Goldfarb proves convergence by defining two algorithms with modified step size conditions. The first method uses a variant of the descent condition,

$$\phi(\alpha) \leq \phi(0) + \mu\left(\alpha g_k^T s_k + \frac{1}{2}\alpha^4 d_k^T H_k d_k\right), \tag{3.20}$$

with Armijo-style backtracking. The second method uses the search function

$$\psi_k(\alpha) = \frac{\phi_k(\alpha) - \phi_k(0)}{g_k^T \Gamma_k(\alpha) + \min\{\frac{1}{2}\Gamma_k(\alpha)^T H_k \Gamma_k(\alpha), 0\}} \tag{3.21}$$

and requires α_k be selected such that $\sigma_1 \leq \psi_k(\alpha_k) \leq \sigma_2$ with $0 < \sigma_1 \leq \sigma_2 < 1$.

It is possible to modify Goldfarb's method to use the arc search conditions of Chapter 2. Note that (3.19), $\liminf_{k\in I} \phi''_k(0) \geq 0$ (Lemma 2.4, part (ii)) and $g_k^T d_k \leq 0$ (Definition 3.2) imply that $\lim_{k\in I} g_k^T d_k = 0$. Then for any subsequence I and fixed tolerance $\epsilon > 0$,

$$|g_k^T d_k| \geq \epsilon |d_k^T H_k d_k| \tag{3.22}$$

implies $\lim_{k\in I} |d_k^T H_k d_k| = 0$. Thus subsequence I would converge to a second-order critical point based on the definition of d_k. In the case where (3.22) does not hold, a Goldfarb arc (3.17) may be replaced by a Forsgren and Murray line (3.7) or a Moré and Sorensen curvilinear arc (3.12) to obtain second-order convergence on all possible subsequences.

Finally, computing the arc-constraint intersection for (3.17) requires finding non-negative real roots of

$$\alpha^2 a_i^T d_k + \alpha a_i^T s_k + a_i^T x_k - b_i = 0.$$

3.4 NEM arcs

This section discusses an arc inspired by the Levenberg-Marquardt algorithm for nonlinear equations. For clarity, the arc is constructed in the context of unconstrained optimization. Linear constraints are handled in Section 3.4.4.

We define a NEM arc as

$$\begin{aligned} \Gamma_k(\alpha) &= Q_k w_k(\alpha) \\ w_k(\alpha) &= -U_k \rho(V_k, \alpha) U_k^T Q_k^T g_k \\ \rho(v, \alpha) &= \frac{\alpha}{1 + \alpha(v - v_{\min,k})}, \end{aligned} \tag{3.23}$$

where $Q_k \in \mathbb{R}^{n \times n_s}$ with $n_s \leq n$, $U_k V_k U_k^T$ is the spectral decomposition of $Q_k^T H_k Q_k$, and $v_{\min,k} = \lambda_{\min}(Q_k^T H_k Q_k)$. The function $\rho(v, \alpha)$ is called the *kernel* and is applied to the diagonal elements of V_k. Unless noted otherwise, Q_k is assumed or constructed to be orthonormal ($Q_k^T Q_k = I$). An NEM arc starts with $\Gamma_k(0) = 0$ as required by the convergence theory. When $H_k \succ 0$, an NEM arc contains the Newton step $p_k = -H_k^{-1} g_k$ if $p_k \in \operatorname{range}(Q_k)$. With $\alpha_N = 1/v_{\min,k}$, $\rho(v, \alpha_N) = 1/v$. Thus, Lemma 3.2 and $p_k \in \operatorname{range}(Q_k)$ imply

$$\Gamma_k(\alpha_N) = -Q_k U_k V_k^{-1} U_k^T Q_k^T g_k = -Q_k (Q_k^T H_k Q_k)^{-1} Q_k^T g_k = p_k.$$

Following the work of Del Gatto [31] on the modified gradient flow algorithm (Section 3.5), the NEM arc may be constructed on a 2-dimensional subspace. The subspace is chosen as

$$S_k = \begin{cases} [g_k \ p_k] & \text{if } H_k \succeq 0 \\ [g_k \ d_k] & \text{if } H_k \not\succeq 0, \end{cases} \tag{3.24}$$

where p_k solves $\min_p \|H_k p + g_k\|_2$ and d_k is a direction of negative curvature satisfying (3.2). The subspace is orthogonalized with $Q_k R_k = \operatorname{qr}(S_k)$.

3.4.1 Derivation

The regularized equation for the Newton step is

$$(H_k + \pi I)s = -g_k. \tag{3.25}$$

An arc can be constructed by considering the solution to (3.25) being parameterized by π:

$$(H_k + \pi I) w_k(\pi) = -g_k. \tag{3.26}$$

Given the spectral decomposition $H_k = U_k V_k U_k^T$, (3.26) can be solved with

$$\begin{aligned} w_k(\pi) &= -U_k \rho(V_k, \pi) U_k^T g_k \\ \rho(v, \pi) &= \frac{1}{v + \pi}. \end{aligned} \tag{3.27}$$

Arc search requires that $\Gamma_k(0) = 0$. However, (3.27) has the property $\lim_{\pi\to\infty} w_k(\pi) = 0$. This issue is resolved with a reparameterization given by

$$\pi(\alpha) = \frac{1}{\alpha} - v_{\min,k}, \tag{3.28}$$

where $v_{\min,k}$ is the minimal eigenvalue of H_k. Note that (3.28) is the solution to the equation $\alpha = \rho(v_{\min,k}, \pi(\alpha))$. Combining (3.28) and (3.27) results in

$$\rho(v, \alpha) = \frac{\alpha}{1 + \alpha(v - v_{\min,k})},$$

the kernel function used in (3.23).

A subspace NEM arc is simply obtained by constructing w_k with $Q_k^T H_k Q_k$ and $Q_k^T g_k$ then "projecting" back to the full space with $\Gamma_k = Q_k w_k$.

3.4.2 Properties

The derivatives and initial value of $\rho(v, \alpha)$ from (3.23) are

$$\begin{aligned} \frac{d}{d\alpha}\rho(v,\alpha) &= \frac{1}{(1+\alpha(v-v_{\min}))^2} & \qquad \frac{d}{d\alpha}\rho(v,\alpha)\Big|_{\alpha=0} &= 1 \\ \frac{d^2}{d\alpha^2}\rho(v,\alpha) &= \frac{-2(v-v_{\min})}{(1+\alpha(v-v_{\min}))^3} & \qquad \frac{d^2}{d\alpha^2}\rho(v,\alpha)\Big|_{\alpha=0} &= -2(v-v_{\min}). \end{aligned}$$

The initial values for the derivatives of $\Gamma_k(\alpha)$ from (3.23) are

$$\begin{aligned} \Gamma_k'(0) &= Q_k w_k'(0) = -Q_k Q_k^T g_k \\ \Gamma_k''(0) &= Q_k w_k''(0) = 2Q_k(Q_k^T H_k Q_k - v_{\min,k} I)Q_k^T g_k. \end{aligned}$$

Lemma 3.3. *If $Q_k^T Q_k = I$, $g_k \neq 0$, and $g_k \in \operatorname{range}(Q_k)$ then an NEM arc defined by (3.23) has the properties*

1. $\Gamma_k(0) = 0$,
2. *if $H_k \succ 0$ and $p_k = -H_k^{-1} g_k \in \operatorname{range}(Q_k)$ then $\Gamma_k(1/v_{\min,k}) = p_k$,*
3. *$\frac{d}{d\alpha}\|\Gamma_k(\alpha)\|_2^2 > 0$ for all $\alpha > 0$ (the norm of the arc is strictly increasing),*
4. *$g_k^T \Gamma_k(\alpha) < 0$ for all $\alpha > 0$ (the arc is always a descent direction),*
5. *$g_k^T \Gamma_k'(\alpha) < 0$ for all $\alpha \geq 0$ (the derivative of the arc is always a descent direction).*

Proof. We denote $v_{i,k}$ as the ith diagonal of V_k and $(U_k^T Q_k^T g_k)_i$ as element i of $U_k^T Q_k^T g_k$.

1. $\rho(v, 0) = 0$ for all v implies $\Gamma_k(0) = 0$.

2. Lemma 3.2 and the definition of ρ from (3.23) imply

$$\Gamma_k(1/v_{\min,k}) = -Q_k U_k V_k^{-1} U_k^T Q_k^T g_k = -Q_k(Q_k^T H_k Q_k)^{-1} Q_k^T g_k = p_k.$$

3. Check $\frac{d}{d\alpha}\|\Gamma_k(\alpha)\|_2^2$:

$$\begin{aligned}\frac{d}{d\alpha}\|\Gamma_k(\alpha)\|_2^2 &= \frac{d}{d\alpha} w_k(\alpha)^T Q_k^T Q_k w_k(\alpha) \\ &= \frac{d}{d\alpha} w_k(\alpha)^T w_k(\alpha) \\ &= \frac{d}{d\alpha} g_k^T Q_k U_k \rho(V_k,\alpha)^2 U_k^T Q_k^T g_k \\ &= 2\sum_{i=1}^{n} \rho(v_{i,k},\alpha)\frac{d}{d\alpha}\rho(v_{i,k},\alpha)(U_k^T Q_k^T g_k)_i^2 > 0,\end{aligned}$$

because $\rho(v,\alpha) > 0$ and $\frac{d}{d\alpha}\rho(v,\alpha) > 0$ for all $\alpha > 0$ and $v > v_{\min,k}$.

4. Check $g_k^T \Gamma_k(\alpha)$:

$$\begin{aligned} g_k^T \Gamma_k(\alpha) &= -g_k^T Q_k U_k \rho(V_k,\alpha) U_k^T Q_k^T g_k \\ &= -\sum_{i=1}^{n} \rho(v_{i,k},\alpha)(U_k^T Q_k^T g_k)_i^2 < 0,\end{aligned}$$

because $\rho(v,\alpha) > 0$ for $\alpha > 0$.

5. Check $g_k^T \Gamma_k'(\alpha)$:

$$\begin{aligned} g_k^T \Gamma_k'(\alpha) &= -g_k^T Q_k U_k \left[\frac{d}{d\alpha}\rho(V_k,\alpha)\right] U_k^T Q_k^T g_k \\ &= -\sum_{i=1}^{n} \frac{d}{d\alpha}\rho(v_{i,k},\alpha)(U_k^T Q_k^T g_k)_i^2 < 0,\end{aligned}$$

because $\frac{d}{d\alpha}\rho(v,\alpha) > 0$ for $\alpha \geq 0$.

□

3.4.3 Convergence

The initial values for the derivatives of the search function for an NEM arc are

$$\begin{aligned}\phi_k'(0) &= g_k^T \Gamma_k'(0) = -g_k^T Q_k Q_k^T g_k \\ \phi_k''(0) &= \Gamma_k'(0)^T H_k \Gamma_k'(0) + g_k^T \Gamma_k''(0) \\ &= g_k^T Q_k Q_k^T H_k Q_k Q_k^T g_k + 2 g_k^T Q_k (Q_k^T H_k Q_k - v_{\min,k} I) Q_k^T g_k.\end{aligned}$$

If $Q_k^T Q_k = I$ and $g_k \in \text{range}(Q_k)$ then by Lemma 3.1 the derivatives simplify to

$$\phi_k'(0) = -g_k^T g_k \tag{3.29}$$

$$\phi_k''(0) = 3g_k^T H_k g_k - 2v_{\min,k} g_k^T g_k. \tag{3.30}$$

Convergence to first-order points is obtained because $\lim_{k\in I} \phi_k'(0) = 0$ (Lemma 2.4, part (i)) and (3.29) imply $\lim_{k\in I} g_k = 0$ where I is any subsequence of unrestricted steps.

The sequences $\{\phi_k'(0)\}_{k\in I}$ and $\{\phi_k''(0)\}_{k\in I}$ from (3.29) and (3.30) do not directly imply $\lim_{k\in I} d_k^T H_k d_k = 0$. Therefore, we present two perturbation methods that may be applied to an NEM arc in order to guarantee second-order convergence.

The first method requires that the subspace matrix Q_k be constructed such that $Q_k^T Q_k = I$ and $d_k \in \text{range}(Q_k)$. The subspace arc function $w_k(\alpha)$ from (3.23) is then redefined with

$$w_k(\alpha) = -U_k \rho(V_k, \alpha) U_k^T Q_k^T (g_k + d_k), \tag{3.31}$$

where d_k satisfies (3.2). After simplification, the initial value for the derivatives of the search function become

$$\phi_k'(0) = -g_k^T (g_k + d_k) \tag{3.32}$$

$$\phi_k''(0) = (g_k + d_k)^T H_k (g_k + d_k) + 2g_k^T H_k (g_k + d_k) - 2v_{\min,k} g_k^T (g_k + d_k). \tag{3.33}$$

Convergence to a second-order critical point follows, because $\lim_{k\in I} g_k = 0$ and $\liminf_{k\in I} \phi_k''(0) \geq 0$ (Lemma 2.4, part (ii)) imply $\lim_{k\in I} d_k^T H_k d_k = 0$, where I is any subsequence with unrestricted steps.

The second method redefines $\Gamma_k(\alpha)$ from (3.23) with

$$\Gamma_k(\alpha) = Q_k w_k(\alpha) + \alpha d_k, \tag{3.34}$$

where d_k satisfies (3.2). The initial value for the derivatives of the search function become

$$\phi_k'(0) = -g_k^T (g_k + d_k) \tag{3.35}$$

$$\phi_k''(0) = (g_k + d_k)^T H_k (g_k + d_k) + 2g_k^T H_k g_k - 2v_{\min,k} g_k^T g_k. \tag{3.36}$$

Convergence to a second-order critical point follows, because $\lim_{k\in I} g_k = 0$ and $\liminf_{k\in I} \phi_k''(0) \geq 0$ (Lemma 2.4, part (ii)) imply $\lim_{k\in I} d_k^T H_k d_k = 0$, where I is any subsequence with unrestricted steps.

If $\epsilon > 0$ is a fixed tolerance and $|g_k^T d_k| \geq \epsilon |d_k^T H_k d_k|$, then $\lim_{k\in I} g_k = 0$ implies $\lim_{k\in I} d_k^T H_k d_k = 0$, where I is any subsequence of unrestricted steps. This indicates that second-order convergence occurs "naturally" if the gradient contains a large enough component in the direction of negative curvature. Therefore, a perturbation only needs to be applied if $|g_k^T d_k| < \epsilon |d_k^T H_k d_k|$. We note that the relative scaling of g_k and d_k becomes an issue if either perturbation (3.31) or (3.34) is used.

However, in practice a perturbation is only applied on a small number of iterations.

3.4.4 Linear constraints

We present two methods using an NEM arc for linearly constrained optimization that differ in the application order of key matrices. First, we review some notation and concepts. Chapter 2 denotes A_k as the matrix of active constraints at the beginning of iteration k and $\bar{A}_k$ as the matrix of constraints that remain active during iteration k. Likewise, Z_k and $\bar{Z}_k$ are the nullspace matrices associated with A_k and $\bar{A}_k$ respectively. A hypothetical arc $\overline{\Gamma}_k$ is constructed to remain on the constraints active at the start of iteration k, i.e., $\overline{\Gamma}_k(\alpha) \in \operatorname{range}(Z_k)$ for all $\alpha \geq 0$. A true search arc Γ_k is constructed to move from constraints deleted in iteration k and remain on $\bar{A}_k$. If constraints are not deleted in iteration k, then $\bar{A}_k = A_k$, $\bar{Z}_k = Z_k$, and $\Gamma_k = \overline{\Gamma}_k$. Thus, $\overline{\Gamma}_k$ may be constructed in the same manner as Γ_k without constraint deletion. Note that hypothetical search arcs are an artifact of the convergence theory and do not need to be implemented in practice.

The QZ method (Procedure 3.1) constructs a NEM arc with a subspace matrix $Q_k \in \mathbb{R}^{n \times n_s}$ such that $\bar{Z}_k\bar{Z}_k^T g_k \in \operatorname{range}(Q_k)$ and $\operatorname{span}(Q_k) \subseteq \operatorname{span}(\bar{Z}_k)$. The definition of w_k differs from (3.23), because g_k is replaced with $\bar{Z}_k\bar{Z}_k^T g_k$. The ZQ method (Procedure 3.2) is equivalent to constructing an unconstrained NEM arc on the reduced variables then "projecting" back to the full space with a product by $\bar{Z}_k$. The methods are named after the order of products in the final definition of Γ_k.

For both methods, the initial derivative of the arc is

$$\Gamma_k'(0) = \begin{cases} \bar{Z}_k\bar{Z}_k^T g_k & \text{if } |g_k^T d_k| \geq \epsilon |d_k^T H_k d_k| \\ \bar{Z}_k\bar{Z}_k^T g_k + d_k & \text{otherwise.} \end{cases} \tag{3.37}$$

Note that d_k remains on the constraints active at the start of iteration k (i.e. $A_k d_k = 0$). Therefore, the arc is initially feasible if

$$a_i^T \bar{Z}_k\bar{Z}_k^T g_k > 0, \tag{3.38}$$

where i is the index of any deleted constraint. The arc satisfies the conditions of Section 2.3.3 if

$$g_k^T \bar{Z}_k\bar{Z}_k^T g_k > g_k^T Z_k Z_k^T g_k, \tag{3.39}$$

which establishes overall convergence of the algorithm. Note that second-order convergence follows from the term d_k in (3.37) and the results from Section 3.4.3.

3.4.5 Constraint intersection

Computing the intersection between an NEM arc and a linear constraint reduces to finding the roots of an order n_s polynomial, where n_s is the dimension of the subspace. For simplicity, we drop indices and use the π-parameterization of (3.27). We assume $a^T x > b$ and search for solutions to

Procedure 3.1 *QZ* method to construct a subspace NEM arc for linear constraints

```
compute $d_k$ to satisfy (3.2)
consider constraint deletion according to rules of Section 2.3.3 to form $\bar{Z}_k$
if $d_k \neq 0$ then
    $S_k \leftarrow [\bar{Z}_k \bar{Z}_k^T g_k \ d_k]$
else
    $p_k \leftarrow \arg\min_p \|\bar{Z}_k^T H_k \bar{Z}_k p + \bar{Z}_k^T g_k\|_2$
    $S_k \leftarrow \bar{Z}_k [\bar{Z}_k^T g_k \ p_k]$
end if
$Q_k R_k \leftarrow \mathrm{qr}(S_k)$
$U_k V_k U_k^T \leftarrow Q_k^T H_k Q_k$ (spectral decomposition)
$\rho(v,\alpha) \leftarrow \frac{\alpha}{1+\alpha(v-v_{\min,k})}$; $v_{\min,k} \leftarrow \lambda_{\min}(Q_k^T H_k Q_k)$
if $d_k = 0$ or $|g_k^T d_k| \geq \epsilon |d_k^T H_k d_k|$ then
    $w_k(\alpha) \leftarrow -U_k \rho(V_k,\alpha) U_k^T Q_k^T \bar{Z}_k \bar{Z}_k^T g_k$
    $\Gamma_k(\alpha) \leftarrow Q_k w_k(\alpha)$
else
    $w_k(\alpha) \leftarrow -U_k \rho(V_k,\alpha) U_k^T Q_k^T (\bar{Z}_k \bar{Z}_k^T g_k + d_k)$
    $\Gamma_k(\alpha) \leftarrow Q_k w_k(\alpha)$
end if
```

Procedure 3.2 *ZQ* method to construct a subspace NEM arc for linear constraints

```
compute $d_k$ to satisfy (3.2)
consider constraint deletion according to rules of Section 2.3.3 to form $\bar{Z}_k$
if $d_k \neq 0$ then
    $S_k \leftarrow [\bar{Z}_k^T g_k \ \bar{Z}_k^T d_k]$
else
    $p_k \leftarrow \arg\min_p \|\bar{Z}_k^T H_k \bar{Z}_k p + \bar{Z}_k^T g_k\|_2$
    $S_k \leftarrow [\bar{Z}_k^T g_k \ p_k]$
end if
$Q_k R_k \leftarrow \mathrm{qr}(S_k)$
$U_k V_k U_k^T \leftarrow Q_k^T \bar{Z}_k^T H_k \bar{Z}_k Q_k$ (spectral decomposition)
$w_k(\alpha) \leftarrow -U_k \rho(V_k,\alpha) U_k^T Q_k^T \bar{Z}_k^T g_k$
$\rho(v,\alpha) \leftarrow \frac{\alpha}{1+\alpha(v-v_{\min,k})}$; $v_{\min,k} \leftarrow \lambda_{\min}(Q_k^T \bar{Z}_k^T H_k \bar{Z}_k Q_k)$
if $d_k = 0$ or $|g_k^T d_k| \geq \epsilon |d_k^T H_k d_k|$ then
    $\Gamma_k(\alpha) \leftarrow \bar{Z}_k Q_k w_k(\alpha)$
else
    $\Gamma_k(\alpha) \leftarrow \bar{Z}_k Q_k w_k(\alpha) + \alpha d_k$
end if
```

$a^T(x + w(\pi)) = b$:

$$\begin{aligned} a^T(x + w(\pi)) &= b \\ a^T w(\pi) &= b - a^T x \\ a^T U\rho(V, \pi)U^T g &= b - a^T x. \end{aligned}$$

Let $\gamma = b - a^T x$ and $\beta_i = (u_i^T a)(u_i^T g)$, where u_i is column i of U. Now we have

$$\begin{aligned} \sum_{i=1}^{n_s} \frac{\beta_i}{v_i + \pi} &= \gamma, \\ \sum_{i=1}^{n_s} \beta_i \prod_{j \neq i} (v_i + \pi) &= \gamma \prod_{j=1}^{n_s} (v_j + \pi). \end{aligned}$$

The final equation is polynomial with order n_s. In the method inspired by Del Gatto [31], $n_s = 2$ and the quadratic formula may be used.

3.4.6 Advantages

The NEM method has several advantages when compared to line and curvilinear search methods. We summarize them here:

- The NEM method does not require a special method to compute s_k when $H_k \not\succ 0$. The Hessian is handled "naturally" in all cases.

- On most iterations, the scaling of d_k is irrelevant because NEM arcs are constructed on orthogonalized subspaces. Perturbations are required to guarantee second-order convergence and are dependent on the scale. However, in practice perturbations are rarely required.

- The vector q_k defined by Forsgren and Murray in [28, Section 5.4] is essentially steepest descent for deleted constraints. NEM arcs defined by Procedures 3.1 and 3.2 first move from the constraints along the steepest descent direction then turn toward the Newton step. Thus, only first-order estimates for the Lagrange multipliers are required, while the algorithm is able to take full advantage of second-order information in selecting the next iterate.

Relative to a line or curvilinear search method, these features come with an $O(nn_s)$ added computational cost, where n_s is the dimension of the subspace. We've shown that a NEM arc may be constructed on a 2-dimensional subspace, and thus the extra work may be considered $O(n)$.

3.5 Modified gradient flow

In the context of unconstrained optimization, Del Gatto [31] defines a *modified Gradient Flow* (MGF) arc as

$$\Gamma_k(\alpha) = Q_k w_k(\alpha) \tag{3.40}$$
$$w_k(\alpha) = -U_k \rho(V_k, \alpha) U_k^T Q_k^T g_k \tag{3.41}$$
$$\rho(v, \alpha) = \begin{cases} -\frac{1}{v}(e^{-vt(\alpha)} - 1) & \text{for } v \neq 0 \\ \alpha & \text{for } v = 0 \end{cases} \tag{3.42}$$
$$t(\alpha) = \frac{-1}{v_{\min,k}} \log(1 - \alpha v_{\min,k}), \tag{3.43}$$

where $Q_k \in \mathbb{R}^{n \times n_s}$ with $n_s \leq n$, $U_k V_k U_k^T$ is the spectral decomposition of $Q_k^T H_k Q_k$, and $v_{\min,k} = \lambda_{\min}(Q_k^T H_k Q_k)$. The MGF and NEM (3.23) arcs share the same definitions of Γ_k and w_k and differ only in the definition of the kernel function ρ. The convergence theory of the two methods is nearly identical, so we omit the details here. In summary, the MGF kernel (3.42) has the properties $\rho(v, 0) = 0$, $\frac{d}{d\alpha}\rho(v, \alpha)\big|_{\alpha=0} = 1$, and $\rho(v, 1/v_{\min,k}) = 1/v$ for $v \geq v_{\min,k}$. Thus, the properties of Section 3.4.2 and convergence results of Section 3.4.3 hold. MGF arcs may also be used for problems with linear constraints in the same manner as Section 3.4.4. Here, we review the derivation of the MGF method and discuss differences with the NEM method.

3.5.1 Derivation

The method of modified gradient flow [3,6,59] defines the search arc as the solution to a linear ODE. For clarity, we discuss the method for unconstrained optimization. The modified gradient flow arc is $\Gamma_k(t) = w_k(t)$, where

$$\begin{aligned} w_k'(t) &= -H_k w(t) - g_k \\ w_k(0) &= 0. \end{aligned} \tag{3.44}$$

Given the spectral decomposition $H_k = U_k V_k U_k^T$, the solution to (3.44) is

$$\begin{aligned} w_k(t) &= -U_k \rho(V_k, t) U_k^T g_k \\ \rho(v, t) &= \begin{cases} -\frac{1}{v}(e^{-vt} - 1) & \text{for } v \neq 0 \\ t & \text{for } v = 0. \end{cases} \end{aligned} \tag{3.45}$$

Here, ρ is called a *kernel function*. The notation $\rho(V_k, t)$ indicates that $\rho(v, t)$ is applied to only the diagonal elements of the eigenvalue matrix V_k. Note that

$$\frac{d}{dt}\rho(v,t) = \rho_t(v,t) = \begin{cases} e^{-vt} & \text{for } v \neq 0 \\ 1 & \text{for } v = 0. \end{cases}$$

Behrman and Del Gatto [3,31] reparameterize (3.45) with

$$t(\alpha) = \begin{cases} \frac{-1}{v_{\min,k}} \log(1 - \alpha v_{\min,k}) & \text{for } v_{\min,k} \neq 0 \\ \alpha & \text{for } v_{\min,k} = 0, \end{cases} \tag{3.46}$$

where $v_{\min,k}$ is the smallest eigenvalue of H_k. This comes from solving for $t(\alpha)$ in the equation $\alpha = \rho(v_{\min,k}, t(\alpha))$. When $v_{\min,k} > 0$ the arc is bounded and the search is over $\alpha \in [0, 1/v_{\min,k}]$. When $v_{\min,k} \leq 0$ the arc is unbounded and the search is over $\alpha \in [0, \infty)$. If $H_k \succ 0$ and $\alpha_N = 1/v_{\min,k}$ then $w_k(t_k(\alpha_N)) = -H_k^{-1} g_k$.

Behrman [3] presents a method that constructs an MGF arc on a subspace spanned by a small number of vectors from the Lanczos process. Del Gatto [31] computes the MGF search arc on a two-dimensional subspace. If $H_k \succeq 0$ then the subspace is chosen as $S_k = [g_k \ p_k]$, where p_k solves $\min_p \|H_k p + g_k\|_2$. If H_k is indefinite, then the subspace is chosen as $S_k = [g_k \ d_k]$. The "projection" onto the subspace requires the tall-skinny QR factorization, $Q_k R_k = \text{qr}(S_k)$. The matrix Q_k has dimensions $n \times 2$. A search arc is computed by solving the linear ODE

$$\begin{aligned} w_k'(t) &= -Q_k^T H_k Q_k w_k(t) - Q_k^T g_k \\ w_k(0) &= 0, \end{aligned} \tag{3.47}$$

then "projecting" back into the full space with $\Gamma_k(t) = Q_k w_k(t)$. The solution to (3.47) requires a spectral decomposition of a 2×2 matrix. A complete MGF arc (3.40) is the solution to (3.47) parameterized by (3.46).

3.5.2 Constraint intersection

Computing the first intersection between an MGF arc and a linear constraint requires finding the smallest real and positive solution to

$$r(t) = \sum_{i=1}^{n_1} \beta_i e^{\nu_i t} + t \sum_{i=1}^{n_2} \gamma_i + \xi = 0, \tag{3.48}$$

where n_1 is the number of nonzero eigenvalues and n_2 is the number of zero eigenvalues of the subspace Hessian $(Q_k^T H_k Q_k)$. The total size of the subspace is $n_1 + n_2$. When $n_1 + n_2 = 2$, it is possible to solve (3.48) with a carefully constructed numerical procedure. Despite some effort we have not found a satisfactory method for the case when $n_1 + n_2 > 2$. A direct search or interpolation scheme would be impractical for large problems.

3.5.3 Comparison to NEM arcs

The only difference between the MGF and NEM methods is the definition of the kernel functions in (3.23) and (3.42). We summarize a few advantages that NEM has over MGF:

- The resulting constraint intersection equation for an NEM arc is a polynomial. For an MGF arc the resulting equation is a sum of exponentials (3.48), which does not appear to have an analytic solution or practical computational routine for subspaces with more than two dimensions.
- The MGF kernel function (3.42) has different expressions for $v \neq 0$ and $v = 0$. Thus the implementation requires a tolerance check on v and procedures to handle both cases. The NEM kernel function is the same for all v.
- The NEM kernel function is qualitatively "nicer", because it does not use an exponential or logarithm.

Chapter 4

ARCOPT

4.1 Preliminaries

ARCOPT is a reduced-gradient method using NEM arcs designed to solve linearly constrained optimization problems of the form

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & l \le \begin{pmatrix} x \\ Ax \end{pmatrix} \le u, \end{array}$$

where $F(x)$ is smooth and A is an $m \times n$ sparse matrix. ARCOPT is influenced by and indebted to MINOS [51]. As in MINOS, the problem is reformulated so that x and A include a full set of slack variables and columns:

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & Ax = 0 \\ & l \le x \le u. \end{array}$$

The primary goal in developing ARCOPT was to adhere to the theoretically defined algorithm from Chapter 2. A few critical deviations were made to account for limited resources and finite precision arithmetic. The implementation is not able to take an infinite number of iterations, thus terminates when approximate optimality conditions are met. Chapter 2 defines an algorithm where all the iterates remain feasible. ARCOPT uses the EXPAND procedure [36], which increases the feasibility tolerance by a small amount each iteration. Variables are allowed to move outside the original bounds (l and u), but must remain feasible with respect to the expanded bounds. This technique is a practical way to reduce the chance of cycling, handle degeneracy, keep key matrices well conditioned, and remove undesirable roots when computing arc-constraint intersections. Even with EXPAND it is still possible for certain matrices to become poorly conditioned. ARCOPT implements a repair procedure from [34, Section 5] to attempt a recovery from these situations.

A fundamental aspect of ARCOPT is a partitioning of the variables and corresponding columns

Table 4.1: Symbols for basis index sets. $\mathcal{B}$, $\mathcal{S}$, $\mathcal{N}_l$, $\mathcal{N}_u$, $\mathcal{N}_f$, and $\mathcal{N}_b$ are disjoint, their union includes all variables $\{1, \dots, m\}$.

symbol	description
$\mathcal{B}$	basic variables
$\mathcal{S}$	superbasic variables
$\mathcal{N}_l$	nonbasic at lower bound
$\mathcal{N}_u$	nonbasic at upper bound
$\mathcal{N}_f$	nonbasic and fixed, $i \in \mathcal{N}_f \Leftrightarrow l_i = u_i$
$\mathcal{N}_b$	nonbasic and between bounds, $l_i < x_i < x_u$
$\mathcal{N}$	union of all nonbasic variables, $\mathcal{N} = \mathcal{N}_l \cup \mathcal{N}_u \cup \mathcal{N}_f \cup \mathcal{N}_b$

of A into several disjoint subsets. *Nonbasic* variables are held fixed, usually at an upper or lower bound. *Superbasic* variables are free to move. *Basic* variables are determined by the linear equations. Symbols for the basis index sets are given in Table 4.1. The linear constraints may be partitioned

$$Ax = \begin{bmatrix} B & S & N \end{bmatrix} \begin{pmatrix} x_{\mathcal{B}} \\ x_{\mathcal{S}} \\ x_{\mathcal{N}} \end{pmatrix}, \tag{4.1}$$

where B is $m \times m$, S is $m \times |\mathcal{S}|$, and N is $m \times n - m - |\mathcal{S}|$. B is called the *basis matrix.* The partition is constructed and maintained so that B has full rank. Given $x_{\mathcal{S}}$ and $x_{\mathcal{N}}$, the basic variables are determined by solving

$$Bx_{\mathcal{B}} = -Sx_{\mathcal{S}} - Nx_{\mathcal{N}}. \tag{4.2}$$

Solves with B and B^T use LU factors from LUSOL [35]. Each iteration may cause a column of B to be replaced with one from S. This is a rank-1 modification and is handled efficiently and stably with LUSOL's Bartels-Golub update of the LU factors. After a certain number of updates, a complete factorization is carried out. Section 4.7 describes products with Z and Z^T, where Z is a matrix whose columns are in null(A).

ARCOPT is composed of several major components. We describe them here in the order invoked by the algorithm:

1. The initialization phase (Section 4.2) processes input and performs all actions required before the main loop.

2. Each iteration of the main loop starts with a call to expand_main (Procedure 4.15) to increase the dynamic feasibility tolerance. After a certain number of iterations the dynamic feasibility tolerance must be reset to its initial value and nonbasic variables must be moved back to their bounds.

3. Each iteration makes a call to phase 1 (Section 4.3) if basic variables are found to be infeasible or

phase 2 (Section 4.4) otherwise. Phase 1 will take a step to minimize the sum of infeasibilities. Phase 2 constructs an NEM arc and takes a step towards optimality. Phase 1 will terminate if the constraints are determined to be infeasible. Phase 2 will terminate if the approximate optimality conditions are met.

4. After each iteration the basis maintenance routine (Section 4.5) is called to handle a change to the basis if a bound was encountered. This may require an update to the basis matrix, which is handled by the factorization routines (Section 4.6).

4.2 Initialization

The initialization phase is responsible for processing solver options, dealing with input data, selecting an initial basis, and performing the initial factorization. The solver options and associated symbols used in this chapter are listed in Table 4.2.

symbol	default	description
δ_D	`1e-6`	dual feasibility tolerance
δ_P	`1e-6`	primal feasibility tolerance
δ_C	`1e-4`	curvature optimality tolerance
δ_M	`0.2`	arc perturbation tolerance
δ_F	`1e-4`	arc search descent parameter
δ_G	`.9`	arc search curvature parameter
δ_V	`1e-7`	initial step size tolerance
δ_R	`2e-4`	regularization parameter
`expfrq`	`10000`	expand reset frequency
δ_A	`.5`	initial EXPAND tolerance
δ_B	`.99`	final EXPAND tolerance
δ_S	`1e-11`	tolerance for near zero numbers

Table 4.2: ARCOPT parameters and default values.

4.2.1 Input

ARCOPT requires:

- Routines to evaluate the objective function $F(x)$ and gradient $g(x)$.
- A routine to evaluate the Hessian $H(x)$, or an operator $H(x, v)$, to evaluate matrix-vector products with the Hessian at x.
- x_0, an initial guess, vector of length n_0.
- A_0, constraint matrix, size $m \times n_0$.
- l, u, lower and upper bounds on variables and constraints with length $n = n_0 + m$.

4.2.2 Initial processing

The input data is processed in the following manner:

1. Compute initial slack variables: $s_0 \leftarrow A_0 x_0$
2. Form augmented constraint matrix: $A \leftarrow [A_0 \;\; -I]$
3. Initialize variable vector x by projecting into bounds:

$$\begin{aligned} x &\leftarrow \begin{pmatrix} x_0 \\ s_0 \end{pmatrix} \\ x &\leftarrow \max(x, l) \\ x &\leftarrow \min(x, u) \end{aligned}$$

4. Call basis_initialize to set the initial basis.
5. Call fac_main to perform the initial factorization.

4.3 Phase 1

Each iteration of Phase 1 carries out the following steps:

1. Check the feasibility of the basic variables. If they are found to be feasible, phase 1 is complete and phase 2 will start in the next iteration. If the basic variables are not feasible, phase 1 continues.
2. Construct a linear objective vector c with length n to minimize the sum of infeasibilities:

$$c_i = \begin{cases} 0 & \text{if } l_i \le x_i \le u_i \\ 1 & \text{if } x_i > u_i \\ -1 & \text{if } x_i < l_i \end{cases}$$

3. Compute the vector of multipliers y by solving $B^T y = c_{\mathcal{B}}$.
4. Compute the residual gradient: $z \leftarrow c - A^T y$
5. Check the optimality conditions. Phase 1 is optimal if

$$\min(x - l, z) \le \delta_D \text{ and } \min(u - x, -z) \le \delta_D.$$

If this occurs, the constraints are not feasible. Here δ_D is the dual feasibility tolerance.

6. Select the nonbasic variable to move. Choose the element of z with largest magnitude and appropriate sign. The index of this variable is denoted k. Set $\sigma \leftarrow -1$ if $z_k < 0$ or $\sigma \leftarrow 1$ if $z_k > 0$.

7. Compute the phase 1 search direction Δx:

$$\Delta x_{\mathcal{N}} \leftarrow 0; \;\; \Delta x_k \leftarrow \sigma; \;\; \Delta x_{\mathcal{B}} \text{ solves } B\Delta x_{\mathcal{B}} = -\sigma a_k,$$

where a_k is column k of A.

8. Compute the maximum possible step size $\bar{\alpha}$ along Δx with the call

$$(\bar{\alpha}, j, \beta, \gamma) \leftarrow \mathsf{expand}(x, \Delta x, l, u, \alpha_{\max}, \delta_E, \delta_T, \delta_S),$$

where $\alpha_{\max}$ is a user defined step size limit and δ_E is the dynamic feasibility tolerance. The EXPAND parameters δ_T and δ_S are described in Section 4.8. If $\gamma = 0$ and $\beta \neq 0$, $\bar{\alpha}$ is limited by a bound on variable j. The lower bound is indicated by $\beta = -1$, while the upper bound is indicated by $\beta = 1$. If $\gamma = 1$ the iteration is deemed degenerate, variable j is made nonbasic, a small step is taken with $x \leftarrow x + \bar{\alpha}\Delta x$, and the algorithm moves to the next iteration.

9. Compute the step size $\alpha \leq \bar{\alpha}$ which removes as many infeasibilities as possible, but does not go any further.

10. Take the step: $x \leftarrow x + \alpha\Delta x$

4.4 Phase 2

Each phase 2 iteration carries out the following steps:

1. Compute the direction of negative curvature. ARCOPT uses Matlab's `eigs` function to compute the eigenvector d corresponding to $v_{\min}$, the minimum eigenvalue Z^THZ. `eigs` uses ARPACK [45] and only requires matrix-vector products with Z^THZ. If all eigenvalues are non-negative, then $d \leftarrow 0$.

2. Compute the vector of multipliers y by solving $B^Ty = g_{\mathcal{B}}$.

3. Compute the residual gradient: $z \leftarrow g - A^Ty$

4. Check the termination conditions. Phase 2 is optimal if

$$\min(x - l, z) \leq \delta_D \text{ and } \min(u - x, -z) \leq \delta_D \text{ and } v_{\min} \geq -\delta_C.$$

5. Consider constraint deletion. This is accomplished by choosing one or more nonbasic variables to make superbasic. Do nothing if a bound limited the step size in the previous iteration. If

the most recent change to the basis was constraint deletion, then more than one constraint may be deleted.

6. Compute the reduced gradient: $g_z \leftarrow Z^T g$

7. Set sign for the direction of negative curvature: $d \leftarrow -d$ if $g_z^T d > 0$

8. Compute the steepest descent direction in the full space: $\Delta x \leftarrow -Z g_z$. If

$$|g_z^T d| < \delta_M |d^T Z^T H Z d|,$$

the steepest descent direction is perturbed with the direction of negative curvature: $\Delta x \leftarrow -Z(g_z + d)$.

9. Compute the maximum possible step size $\bar{\alpha}$ along Δx with the call

$$(\bar{\alpha}, j, \beta, \gamma) \leftarrow \mathsf{expand}(x, \Delta x, l, u, \alpha_{\max}, \delta_E, \delta_T, \delta_S),$$

where $\alpha_{\max}$ is a user defined step size limit and δ_E is the dynamic feasibility tolerance. The EXPAND parameters δ_T and δ_S are described in Section 4.8. If $\gamma = 0$ and $\beta \neq 0$, $\bar{\alpha}$ is limited by a bound on variable j. The lower bound is indicated by $\beta = -1$, while the upper bound is indicated by $\beta = 1$. If $\gamma = 1$, then the iteration is deemed degenerate, variable j is made nonbasic, a small step is taken with $x \leftarrow x + \bar{\alpha}\Delta x$, and the algorithm moves to the next iteration.

10. If $v_{\min} \geq -\delta_C$, compute the regularized Newton direction p with

$$(Z^T H Z + \delta_R I)p = -g_z,$$

where δ_R is the regularization parameter. This is accomplished with Matlab's `pcg` or `minres`.

11. Construct the arc $\Gamma(\alpha)$. If no direction of negative curvature exists ($v_{\min} \geq -\delta_C$), the subspace is chosen to be $[g_z \ p]$. If a direction of negative curvature exists ($v_{\min} < \delta_C$), then the arc subspace is chosen to be $[g_z \ d]$. If $|g_z^T d| \leq \delta_M |d^T Z^T H Z d|$ the arc must be perturbed so that $\Gamma'(0) = -Z(g_z + d)$.

12. Compute the maximum step size along the arc:

$$\bar{\alpha} \leftarrow \max \ \alpha \ \text{ such that } \ l - \delta_E \leq x + \Gamma(\alpha) \leq u + \delta_E.$$

This is done with the call

$$(\bar{\alpha}, j, \beta) \leftarrow \mathsf{arctest}(x, \Gamma, l, u, \alpha_{\max}, \delta_E),$$

where $\alpha_{\max}$ is a user defined step size limit and δ_E is the dynamic feasibility tolerance. If $\beta \neq 0$ then j is the index of the limiting bound. The lower bound is indicated by $\beta = -1$, while the upper bound is indicated by $\beta = 1$.

13. Compute the initial step size. If the reduced Hessian is positive definite ($v_{\min} \geq \delta_V$), the initial step size is $\alpha_0 = 1/v_{\min}$. If the reduced Hessian is positive semi-definite ($|v_{\min}| < \delta_V$), the initial step size is $\alpha_0 = 1$. If the reduced Hessian is indefinite ($v_{\min} \leq \delta_V$), the initial step size is $\alpha_0 = -1/v_{\min}$. These choices were used in both [3] and [31].

14. Search along arc for an acceptable step size. The search function is $\phi(\alpha) = F(x + \Gamma(\alpha))$. First, compute values for $\phi'(0)$ and $\phi''(0)$. Second, find α that satisfies

$$\begin{aligned} \phi(\alpha) &\leq \phi(0) + \delta_F \left(\phi'(0)\alpha + \tfrac{1}{2}\min\{\phi''(0), 0\}\alpha^2\right) \\ |\phi'(\alpha)| &\leq \delta_G |\phi'(0) + \min\{\phi''(0), 0\}\alpha|. \end{aligned}$$

15. Take the step: $x \leftarrow x + \Gamma(\alpha)$

4.5 Basis maintenance

Basis maintenance is performed at the end of each phase 1 or phase 2 iteration. If the step was limited by a bound, the corresponding variable must be moved to the appropriate nonbasic set. If the limiting variable was basic, its position in the basis must be replaced with a superbasic variable. Changes to the basis may call for an update or refactorization of B. If the new basis matrix is found to be ill-conditioned or rank deficient, then factorization repair is invoked. These steps are detailed in the following routines:

- basis_initialize (Procedure 4.1): perform initial partitioning of variables.
- basis_main (Procedure 4.2): main call for basis maintenance. Responsible for moving a limiting variable to the appropriate nonbasic set and updating the factorization if needed.
- basis_activate (Procedure 4.3): move limiting variable to appropriate nonbasic set.
- basis_select (Procedure 4.4): select an appropriate superbasic variable to move to basic set.
- vmap (Procedure 4.6): map from basis matrix column index to variable index.
- bmap (Procedure 4.5): map from basis variable index to basis matrix column index.

Procedure 4.1 basis_initialize: perform the initial partition of variables into basis sets.

```
for $i = 1$ to $n$ do
  if $l_i = u_i$ then
    add $i$ to $\mathcal{N}_f$, $x_i$ is nonbasic and fixed
  else if $x_i = l_i$ then
    add $i$ to $\mathcal{N}_l$, $x_i$ is nonbasic at lower bound
  else if $x_i = u_i$ then
    add $i$ to $\mathcal{N}_u$, $x_i$ is nonbasic at upper bound
  else
    add $i$ to $\mathcal{S}$, $x_i$ is superbasic
  end if
end for
if crash = firstm then
  $\mathcal{B} \leftarrow \{1, \dots, m\}$, make first $m$ variables basic
  remove $\{1, \dots, m\}$ from other basis sets
else
  $\mathcal{B} \leftarrow \{n - m + 1, \dots, n\}$, make last $m$ (slack) variables basic
  remove $\{n - m + 1, \dots, n\}$ from other basis sets
end if
```

Procedure 4.2 basis_main(j, β): main entry point to basis routines. Responsible for moving a limiting variable to the appropriate nonbasic set and updating the factorization if needed.

```
Require: $j \in \{0\} \cup \mathcal{B} \cup \mathcal{S}$ is index of limiting basic or superbasic variable
  // $j = 0$ indicates no variable is limited by a bound
Require: $\beta \in \{-1, 0, 1\}$ indicates upper or lower limit
  // $\beta = -1$ indicates lower bound limit
  // $\beta = 0$ indicates no variable is limited by a bound
  // $\beta = 1$ indicates upper bound limit
Ensure: basis index sets and factorization are updated
  if $\beta \neq 0$ and $j \in \mathcal{S}$ then
    // superbasic variable $j$ has hit bound
    call basis_activate$(j, \beta)$ to make $j$ nonbasic
  else if $\beta \neq 0$ and $j \in \mathcal{B}$ then
    // basic variable $j$ has hit bound, need to find superbasic variable to replace
    $i \leftarrow$ basis_select$(j)$
    move variable $i$ to $\mathcal{B}$
    call basis_activate$(j, \beta)$ to make $j$ nonbasic
    call fac_update$(i, j)$ to replace column vmap$(j)$ of $B$ with column $i$ of $A$
    // fac_update may call fac_main to refactorize or repair $B$
  end if
```

Procedure 4.3 basis_activate(j, β): move limiting variable j to appropriate nonbasic set.

```
Require: $j \in \mathcal{B} \cup \mathcal{S}$ is index of limiting variable
Require: $\beta \in \{-1, 1\}$ indicates lower or upper limit, respectively
Ensure: $j$ is made nonbasic
  if $\beta = -1$ then
    move $j$ to $\mathcal{N}_l$
  else if $\beta = 1$ then
    move $j$ to $\mathcal{N}_u$
  end if
```

Procedure 4.4 $i \leftarrow \mathsf{basis\_select}(j)$: select a superbasic variable to become basic. The method is from [51, p. 53].

```
Require: $j \in \{1, m\}$, index of basic variable to become nonbasic
Require: $|\mathcal{S}| \geq 1$, there must be at least one superbasic variable
Ensure: $i$ is index of superbasic variable to become basic
  $k \leftarrow \mathsf{vmap}(j)$ // index of column in $B$ corresponding to $j$
  // find largest available pivot, $v_{\max}$
  $u$ solves $B^T u = e_k$
  $v \leftarrow |S^T u|$
  $v_{\max} \leftarrow \max(v)$
  // compute minimum distance to bound for each superbasic
  $d_k \leftarrow \min\{|x_k - l_k|, |x_k - u_k|\}$ for $k \in \mathcal{S}$
  $i \leftarrow \arg\max_k\{d_k$ with $v_k \geq 0.1 v_{\max}$ and $k \in \mathcal{S}\}$
  // $i$ is index of variable furthest from bound with $v_i \geq 0.1 v_{\max}$
```

Procedure 4.5 $j \leftarrow \mathsf{bmap}(i)$: map from basis matrix column index to variable index

```
Require: $i \in \{1, \ldots, m\}$ is the column index of $B$
  return $j \in \mathcal{B}$, the index of basic variable corresponding to column $i$ of $B$
```

Procedure 4.6 $i \leftarrow \mathsf{vmap}(j)$: map from basis variable index to basis matrix column index

```
Require: $j \in \mathcal{B}$ is an index of a basic variable
  return the index of basis matrix column corresponding to variable $j$
```

4.6 Factorization

ARCOPT uses LUSOL [35] to compute and update sparse LU factors of the basis matrix B. If B is found to be ill-conditioned or rank deficient, ARCOPT uses a repair procedure presented by Gill, Murray, Saunders, and Wright in their paper on SNOPT [34, Section 5]. The factorization and repair routines are summarized here:

- fac_main (Procedure 4.7) main controller for the basis factorization. It first attempts to compute LU factors for B. If the basis is found to be ill-conditioned or rank deficient, then basis repair is invoked.
- fac_update (Procedure 4.8): update LU factors to reflect a column replacement in B.
- fac_BS (Procedure 4.10): attempt to find a better conditioned basis by swapping basic and superbasic variables.
- fac_BR (Procedure 4.9): construct a well conditioned basis by replacing dependent columns of B with appropriate columns corresponding to slack variables.
- fac_repair (Procedure 4.11): help routine called by fac_BS to select appropriate slack variables for inclusion in the basis.

Procedure 4.7 fac_main: factorize B with LUSOL using threshold partial pivoting (TPP). If B is singular, call fac_BS to attempt to fix the basis by swapping in superbasic variables. If basis is still rank deficient, call fac_BR to replace dependent columns with appropriate ones corresponding to slack variables.

```
Require: function nsing($U$) counts number of apparent singularities in $U$
  $r \leftarrow$ false // flag to indicate a basis repair call
  $(L, U, p, q) \leftarrow$ LUSOL($B$, TPP)
  if nsing($U$) $> 0$ and $|\mathcal{S}| > 0$ then
    // basis is singular, attempt to replace dependent columns with superbasics
    call fac_BS
    $r \leftarrow$ true
  end if
  if nsing($U$) $> 0$ then
    // basis is singular, replace dependent columns with slacks
    call fac_BR
    $r \leftarrow$ true
  end if
  if $r$ is true then
    // refactorize because a basis repair routine was called
    $(L, U, p, q) \leftarrow$ LUSOL($B$, TPP)
  end if
  store $L$, $U$, $p$, and $q$ for subsequent solves and updating
```

Procedure 4.8 fac_update(i,j): update factorization to replace column in B corresponding to variable j with column in A corresponding to variable i. Call fac_main if the update routine reports a singular result. Note that LUSOL maintains updates to L factors in product form; we ignore that detail here.

```
$j \leftarrow \mathsf{vmap}(j)$ // get column index of $B$ corresponding to variable $j$
$(L,U,p,q,r) \leftarrow \mathsf{LUSOL\_REPCOL}(L,U,p,q,j,a_i)$ // $a_i$ is column $i$ of $A$
if $r$ is true then
  // singularity detected, initiate repair procedure
  call fac_main
end if
```

Procedure 4.9 fac_BS: factorize $[B\ S]^T$ with LUSOL's threshold rook pivoting (TRP) in order to find a full rank or better conditioned set of basis columns without changing the state of non-basic variables. The LU factors are not stored and a subsequent factorization is required for solves.

```
$(L,U,p,q) \leftarrow \mathsf{LUSOL}([B\ S]^T, \mathsf{TRP})$
// move first $m$ pivot rows to $\mathcal{B}$
for $i = 1$ to $m$ do
  move $p_i$ to $\mathcal{B}$
end for
// move remaining $|\mathcal{S}|$ pivot rows to $\mathcal{S}$
for $i = m+1$ to $m + |\mathcal{S}|$ do
  move $p_i$ to $\mathcal{S}$
end for
```

Procedure 4.10 fac_BR: factorize B with LUSOL's threshold rook pivoting (TRP) to find dependent columns. Dependent columns are replaced with identity columns corresponding to slack variables in the fac_repair method. There is no guarantee that one factorization and subsequent call to fac_repair will produce a nonsingular basis. Thus, the method will repeat until a full rank basis is produced. This is always possible, because the entire basis matrix could replaced with $-I$. For efficiency the method throws the LU factors away. A subsequent factorization is needed for solves.

```
repeat
  $(L,U,p,q) \leftarrow \mathsf{LUSOL}(B, \mathsf{TRP})$
  if $\mathsf{nsing}(U) > 0$ then
    call fac_repair to replace dependent columns of $B$ with slacks
  end if
until $\mathsf{nsing}(U) = 0$
```

Procedure 4.11 fac_repair: repair B by making variables corresponding to dependent columns nonbasic and appropriate slack variables basic.

```
Require: $p$ and $q$ are row and column pivot vectors from LUSOL
Require: depcol($j$) returns true if column $j$ of $B$ is dependent
  for $i = 1$ to $m$ do
    $j \leftarrow q_i$ // get appropriate column index
    if depcol($j$) then
      $k \leftarrow$ bmap($j$)
      // column in $B$ corresponding to variable $k$ is dependent
      // make variable $k$ nonbasic
      if $x_k \leq l_k$ then
        move $k$ to $\mathcal{N}_l$
      else if $x_k \geq u_k$ then
        move $k$ to $\mathcal{N}_u$
      else if $l_k = u_k$ then
        move $k$ to $\mathcal{N}_f$
      else
        move $k$ to $\mathcal{N}_b$
      end if
      // move slack variable into basis to repair $B$
      move variable with index $n - m + p_k$ to $\mathcal{B}$
    end if
  end for
```

4.7 Products with Z and Z^T

ARCOPT requires the ability to compute products with a nullspace matrix Z. If the variables are partitioned such that $A = [B\ S\ N]$ (4.1) a nullspace matrix can be constructed as the $m \times |\mathcal{S}|$ matrix

$$Z = \begin{bmatrix} -B^{-1}S \\ I \\ 0 \end{bmatrix}. \tag{4.3}$$

The nullspace matrix is not constructed but used as an operator to compute products with Z (Procedure 4.12) and Z^T (Procedure 4.13).

Procedure 4.12 $v \leftarrow Zu$

```
**Require:** $u \in \mathbb{R}^{|\mathcal{S}|}$, $B$ is full rank
**Ensure:** $v \in \text{null}(A)$
  $v_{\mathcal{S}} \leftarrow u$
  $v_{\mathcal{N}} \leftarrow 0$
  $v_{\mathcal{B}}$ solves $Bv_{\mathcal{B}} = -Su$
```

Procedure 4.13 $v \leftarrow Z^T u$

```
**Require:** $u \in \mathbb{R}^n$, $B$ is full rank
**Ensure:** $v \in \text{range}(Z^T)$
  $u_1$ solves $B^T u_1 = u_{\mathcal{B}}$
  $v \leftarrow -S^T u_1 + u_{\mathcal{S}}$
```

4.8 Expand

Gill, Murray, Saunders, and Wright presented the EXPAND procedure [36] to prevent cycling in active set methods for linearly constrained optimization. ARCOPT's initialization procedure calls expand_init (Procedure 4.14) to set the dynamic feasibility tolerance δ_E and the growth parameter δ_T. Before each phase 1 or phase 2 iteration, expand_update (Procedure 4.15) is called to increase the dynamic feasibility tolerance ($\delta_E \leftarrow \delta_E + \delta_T$). If δ_E has grown too large, then expand_reset (Procedure 4.16) is called to set δ_E to its initial value and move nonbasic variables back to their bounds. Phase 1 is invoked if basic variables become infeasible.

The main EXPAND routines are used to determine the largest step size and select a "good" limiting constraint. At the lowest level, step (Procedure 4.19) computes the step size to a bound along a line for a single variable. Note that step uses parameter δ_S as a tolerance on near zero values. Next, ratio_test (Procedure 4.18) computes the maximum step size such that all variables remain feasible. Finally, expand (Procedure 4.17) computes the maximum step size along a line to a constraint with a large "pivot" value such that all variables remain feasible with respect to the expanded bounds. Small "pivot" values typically lead to ill-conditioned basis matrices [36, p. 441].

An iteration is deemed degenerate if the step size to the nearest bound is too small. This occurs when multiple constraints are encountered in the same iteration. In this situation, the expand routine enforces a small positive step size such that all variables remain feasible with respect to the expanded bounds. A degeneracy flag is set that signals ARCOPT to take the step, perform basis updates, then go to the next iteration. In phase 2, ARCOPT checks for degeneracy by calling expand on $x + \alpha\Gamma'(0)$, which does not involve any work to compute arc-constraint intersections. In fact, the arc is only constructed if the iteration is deemed non-degenerate.

Procedure 4.14 expand_init: initialize dynamic feasibility tolerance and growth parameter.

```
$\delta_E \leftarrow \delta_A \cdot \delta_P$
$\delta_T \leftarrow \frac{(\delta_B - \delta_A)\delta_P}{\texttt{expfrq}}$
```

Procedure 4.15 expand_update: increase dynamic feasibility tolerance, reset if needed.

```
$\delta_E \leftarrow \delta_E + \delta_T$
if $\delta_E \geq \delta_P$ then
    call expand_reset to reset dynamic tolerance
end if
```

Procedure 4.16 expand_reset: reset dynamic tolerance, bring infeasible variables back to bounds.

```
move infeasible nonbasic variables back to bounds
call fac_main to refactorize matrix
call exp_init to reset dynamic feasibility tolerance
recompute basic variables
if basic variables are found to be infeasible, return to phase 1
```

Procedure 4.17 $(\alpha, j, \beta, \gamma) \leftarrow$ expand$(x, \Delta x, l, u, \alpha_{\max}, \delta_E, \delta_T, \delta_S)$: find the largest step size to a bound associated with a large pivot such that all variables remain feasible with respect to the expanded bounds. If the computed step size is too small, then set the degeneracy flag and return a small positive step size.

```
Require: $x, \Delta x, u, l \in \mathbb{R}^n$, $\alpha_{\max}, \delta_E, \delta_T, \delta_S \in \mathbb{R}_+$
Require: $l - \delta_E < x < u + \delta_E$
Ensure: $\alpha \in (0, \alpha_{max}], j \in \{0, \ldots, n\}, \beta \in \{-1, 0, 1\}, \gamma \in \{0, 1\}$
  $\alpha \leftarrow \alpha_{\max};\ j \leftarrow 0;\ \beta \leftarrow 0;\ \gamma \leftarrow 0;\ p \leftarrow 0$
  $(\alpha_1, j_1, \beta_1) \leftarrow$ ratio_test$(x, \Delta x, l - \delta_E, u + \delta_E, \alpha_{\max}, \delta_S)$
  for $i = 1$ to $n$ do
    $(\alpha_2, \beta_2) \leftarrow$ step$(x_i, \Delta x_i, l_i, u_i, \alpha_{\max}, \delta_S)$
    if $\alpha_2 \leq \alpha_1$ and $|\Delta x_i| > p$ then
      $\alpha \leftarrow \alpha_2;\ j \leftarrow i;\ \beta \leftarrow \beta_2;\ p \leftarrow |\Delta x_i|$
    end if
  end for
  $\alpha_{\min} \leftarrow \delta_T / p$
  if $\alpha \leq \alpha_{\min}$ then
    $\alpha \leftarrow \alpha_{\min};\ \gamma \leftarrow 1$
  end if
```

Procedure 4.18 $(\alpha, j, \beta) \leftarrow$ ratio_test$(x, \Delta x, l, u, \alpha_{\max}, \delta_S)$: find the largest step size to a bound such that all variables remain feasible.

```
Require: $x, \Delta x, u, l \in \mathbb{R}^n$, $\alpha_{\max}, \delta_S \in \mathbb{R}_+$
Require: $l < x < u$
Ensure: $\alpha \in (0, \alpha_{max}], j \in \{0, \ldots, n\}, \beta \in \{-1, 0, 1\}$
  $\alpha \leftarrow \alpha_{max};\ j \leftarrow 0;\ \beta \leftarrow 0$
  for $i = 1$ to $n$ do
    $(\alpha_1, \beta_1) \leftarrow$ step$(x_i, \Delta x_i, l_i, u_i, \alpha_{\max}, \delta_S)$
    if $\alpha_1 < \alpha$ then
      $\alpha \leftarrow \alpha_1;\ j \leftarrow i;\ \beta \leftarrow \beta_1$
    end if
  end for
```

Procedure 4.19 $(\alpha, \beta) \leftarrow \mathsf{step}(x, \Delta x, l, u, \alpha_{\max}, \delta_S)$: find the largest step size to a bound for a single variable.

```
Require: $x, \Delta x, u, l \in \mathbb{R}$, $\alpha_{\max}, \delta_S \in \mathbb{R}_+$
Ensure: $\alpha \in (0, \alpha_{max}], \beta \in \{-1, 0, 1\}$
  $\alpha \leftarrow \alpha_{max};\ \beta \leftarrow 0$
  if $\Delta x < -\delta_S$ and $l > -\infty$ then
    $\alpha \leftarrow (l - x)/\Delta x;\ \beta \leftarrow -1$
  end if
  if $\Delta x > \delta_S$ and $u < \infty$ then
    $\alpha \leftarrow (u - x)/\Delta x;\ \beta \leftarrow 1$
  end if
  if $\alpha > \alpha_{\max}$ then
    $\alpha \leftarrow \alpha_{\max};\ \beta \leftarrow 0$
  end if
```

4.9 Arc-constraint intersection

In phase 2, ARCOPT computes the maximum step size along an arc with arctest (Procedure 4.20), which computes

$$\bar{\alpha} = \max\{\alpha \text{ such that } l - \delta_E \leq x + \Gamma(\alpha) \leq u + \delta_E\}$$

and also returns the index of the limiting variable. The routine arcbound (Procedure 4.22) finds the smallest non-negative real root of the nonlinear equation arising from the intersection of an arc and a bound for a single variable. The routine arcstep (Procedure 4.21) applies arcbound to both upper and lower bounds for a single variable. Note that arcstep and arcbound assume that the input variable is strictly feasible with respect to the input bounds. The expanding feasibility tolerance ensures this property for all variables.

Procedure 4.20 $(\alpha, j, \beta) \leftarrow$ arctest$(x, \Gamma, l, u, \alpha_{\max}, \delta)$: find the largest step size along an arc such that all variables remain feasible with respect to expanded bounds.

```
Require: $x, l, u \in \mathbb{R}^n$, $\alpha_{\max}, \delta \in \mathbb{R}_+$, $\Gamma \in C[0, \alpha_{\max}] : \mathbb{R} \mapsto \mathbb{R}^n$
Require: $l - \delta < x + \Gamma(0) < u + \delta$
Ensure: $\alpha \in (0, \alpha_{\max}]$, $j \in [0, n]$, $\beta \in \{-1, 0, +1\}$
  $\alpha \leftarrow \alpha_{\max}$; $j \leftarrow 0$; $b \leftarrow 0$
  for $i = 1$ to $n$ do
    $(\alpha_i, \beta_i) \leftarrow$ arcstep$(x_i, \Gamma_i, l_i - \delta, u_i + \delta, \alpha_{\max})$
    if $\alpha_i \leq \alpha$ and $\beta_i \neq 0$ then
      $\alpha \leftarrow \alpha_i$; $j \leftarrow i$; $\beta \leftarrow \beta_i$
    end if
  end for
```

Procedure 4.21 $(\alpha, \beta) \leftarrow$ arcstep$(x, \Gamma, l, u, \alpha_{\max})$: find the largest step size along an arc for a single variable.

```
Require: $x, l, u \in \mathbb{R}$, $\alpha_{\max} \in \mathbb{R}_+$, $\Gamma \in C[0, \alpha_{\max}] : \mathbb{R} \mapsto \mathbb{R}$
Require: $l < x + \Gamma(0) < u$
Ensure: $\alpha \in (0, \alpha_{\max}], \beta \in \{-1, 0, 1\}$
  $(\alpha_l, \beta_l) \leftarrow$ arcbound$(x, \Gamma, l, \alpha_{\max})$
  $(\alpha_u, \beta_u) \leftarrow$ arcbound$(-x, -\Gamma, -u, \alpha_{\max})$
  if $\alpha_l < \alpha_u$ then
    $\alpha \leftarrow \alpha_l$; $\beta \leftarrow -\beta_l$
  else
    $\alpha \leftarrow \alpha_u$; $\beta \leftarrow \beta_u$
  end if
```

Procedure 4.22 $(\alpha, \beta) \leftarrow \mathsf{arcbound}(x, \Gamma, l, \alpha_{\max})$: compute the first point of intersection between an arc and a bound for a single variable.

```
**Require:** $x, l \in \mathbb{R}$, $\alpha_{\max} \in \mathbb{R}_+$, $\Gamma \in C[0, \alpha_{\max}] : \mathbb{R} \mapsto \mathbb{R}$
**Require:** $l < x + \Gamma(0)$
**Ensure:** $\alpha \in (0, \alpha_{\max}], \beta \in \{0, 1\}$
  $\alpha \leftarrow \alpha_{\max}$; $\beta \leftarrow 0$
  **if** $l > -\infty$ **then**
    $r \leftarrow \mathsf{roots}(\Gamma + (x - l), [0, \alpha_{\max}])$
    $r_{\min} \leftarrow \mathsf{min}(r)$
    **if** $r_{\min}$ exists **then**
      $\alpha \leftarrow r_{\min}$; $\beta \leftarrow 1$
    **end if**
  **end if**
```

Chapter 5

Experiments

5.1 Preliminaries

This chapter documents the following numerical experiments:

- A comparison between ARCOPT and IPOPT on a continuous formulation of the Hamiltonian cycle problem.
- A comparison between ARCOPT, IPOPT, and SNOPT on problems from the CUTEr test set.
- A comparison of the BFGS and SR1 quasi-Newton updates in an arc search code.

We begin by discussing existing solvers and a method of comparing performance.

5.1.1 SNOPT

SNOPT is an active-set SQP method for large-scale sparse nonlinear optimization by Gill, Murray, and Saunders [34]. The SNOPT algorithm does not use second derivatives and thus cannot guarantee convergence to second-order critical points. However, since it is a descent method, SNOPT finds a minimizer most of the time. SNOPT maintains a limited-memory quasi-Newton approximation of the Hessian. The software is known to be robust and efficient, which makes it an attractive candidate for comparison.

5.1.2 IPOPT

IPOPT is an interior point code by Wächter and Biegler [60] using a filter-based line search for nonlinearly constrained optimization. IPOPT is distributed as open source software and is able to use second derivatives. When the Hessian is indefinite, IPOPT computes a search direction from

$$(H + \lambda I)s = -g.$$

The method to select λ is fully described in [60, Section 3.1]. In summary, if H is found to be indefinite, IPOPT sets $\lambda \leftarrow \lambda_0$, where λ_0 is determined from a user parameter or a previous iteration. If $H + \lambda I$ is found to be indefinite then λ is increased by a factor δ such that $\lambda \leftarrow \delta\lambda$ and the process is repeated. The default setting is $\delta = 8$. Between iterations, the initial trial value λ_0 is decreased.

IPOPT does not explicitly compute or use directions of negative curvature. Convergence to second-order critical points is not guaranteed, but often observed in practice.

5.1.3 Other solvers

Table 5.1 lists related solvers that are available for use on NEOS [18]. For each solver we list the problem type, basic method, ability to accept AMPL [29] models, and use of second derivatives. All solvers require smooth objective functions.

None of the methods listed guarantees convergence to second-order critical points. We did a simple test on the solvers in Table 5.1 that accept AMPL models. The problems were

$$\text{minimize } F_1(x,y) = x^2 - 3y^2 + y^4 \quad \text{and} \tag{5.1}$$

$$\text{minimize } F_2(x,y) = x^2 - y^2 \ \text{ subject to } \ -2 \le x, y \le 2, \tag{5.2}$$

which have saddle points at $(0,0)$. Problem (5.1) has (global) minimizers at $(x,y) = (0, \pm\sqrt{3/2})$. Problem (5.2) has (global) minimizers at $(x,y) = (0, \pm 2)$. When started at $(x_0, y_0) = (1, 0)$ all solvers terminate at the saddle point, with the exception of LOQO which found a minimizer for (5.1) and the saddle point for (5.2). It should be noted that the initial point is very special, because it lies in the positive definite subspace of the Hessian. If an algorithm does nothing to move off this space it will likely converge to the saddle point. For these problems, all solvers converge to a minimizer if (x_0, y_0) is selected to contain a large enough component in the span of $(0, 1)$. This exercise demonstrates that convergence to second-order critical points is a feature missing from *all* solvers available to the community despite many of those solvers taking advantage of second derivatives.

5.1.4 Performance profiles

Dolan and Moré developed performance profiles to compare optimization software on a set of test problems [19, 20]. We briefly describe the method here and use it throughout this chapter. Denote the set of solvers $\mathcal{S}$ and the set of test problems $\mathcal{P}$. The metric $t_{p,s}$ indicates the performance of solver $s \in \mathcal{S}$ on problem $p \in \mathcal{P}$. The metric could be solution time, number of function evaluations, or solution quality. The best performance on any problem is given by $t_{p,\min} = \min\{t_{p,s} : s \in \mathcal{S}\}$. The ratio $r_{p,s} = t_{p,s}/t_{p,\min}$ indicates the relative performance of solver s on p. We see $r_{p,s} = 1$ if s achieved the best observed performance on p and $r_{p,s} > 1$ otherwise. If solver s failed on p, then $r_{p,s}$ can be set to a sufficiently large number. Finally, performance profiles are plots of the function

$$f_s(\sigma) = \frac{|\{p : r_{p,s} \le \sigma\}|}{|\mathcal{P}|},$$

Table 5.1: List of solvers available on NEOS [18].

solver	type	method	AMPL?	$\nabla^2 F$?	
L-BFGS-B	BC	LS	*		[13]
TRON	BC	TR		*	[46]
CONOPT	NC	LS	*		[21]
filter	NC	TR	*	*	[26]
KNITRO	NC	LS/TR	*	*	[12]
LANCELOT	NC	TR	*	*	[17]
LOQO	NC	LS	*	*	[58]
LRAMBO	NC	LS		*	
MINOS	NC	LS	*		[51]
PATHNLP	NC	?		*	
SNOPT	NC	LS	*		[34]
NMTR	UC	TR		*	[49]

UC	unconstrained
BC	bound constrained
NC	nonlinearly constrained
LS	line search
TR	trust region

which is the fraction of problems that s was able to solve with performance ratio at most σ. The profile functions $f_s(\sigma)$ are drawn for all $s \in \mathcal{S}$ on the same plot. Solvers with greater area under the profile curve exhibit better relative performance on the test set.

A possible criticism of this performance profile is that it may give misleading results if applied to a set of problems that vary widely in difficulty. Differences in the performance metric on easy problems (small $t_{p,\min}$) will have a much larger impact on relative performance when compared to performance differences on hard problems (large $t_{p,\min}$). Thus, the overall profile for a solver that does well on hard problems may look poor if it performs marginally worse on easy problems. In our situation this is not an issue. In the next section we compare solvers on different instances of the same problem. In the following sections we use the CUTEr test set. In both cases, the problems do not have a large variation in difficulty.

5.2 Hamiltonian cycle problem (HCP)

A Hamiltonian cycle (HC) is a path through an undirected graph that follows edges to visit each node exactly once and returns to the start. Finding such cycles in a graph is known as the Hamiltonian cycle problem and is one of Karp's 21 problems shown to be NP-complete [44]. It is simple to check whether a given cycle is Hamiltonian. However, there is no known algorithm guaranteed to find an HC, or report that one does not exist, in time proportional to a polynomial function of the number of nodes and edges.

Ejov, Filar, Murray, and Nguyen derived an interesting continuous formulation of HCP [22]. Consider a graph with n nodes and m edges. The variables in the problem are weights of an adjacency matrix, denoted $P(x)$. Each undirected edge is made of two directed edges connecting the same nodes in opposite directions. There is one variable per directed edge. Therefore, x has $2m$ elements. The optimization problem is

$$\begin{array}{ll} \underset{x}{\text{minimize}} & G(x) = \det\left(I - P(x) + \frac{1}{N} e e^T\right) \\ \text{subject to} & P(x)e = e \\ & x \geq 0, \end{array} \tag{5.3}$$

where e is a vector of ones. If the graph has a Hamiltonian cycle, then it can be shown that x^* is a global minimizer for (5.3) if $G(x^*) = -N$. The elements of x^* are 0 or 1, where the 1's correspond to edges that are included in the cycle. Global minimizers also result in a doubly stochastic adjacency matrix $P(x^*)$, which satisfies the additional constraint $P(x^*)^T e = e$. Filar, Haythorpe, and Murray derived an efficient method to evaluate the first and second derivatives of $G(x)$ [25].

It turns out that using second derivatives and directions of negative curvature is particularly important in finding HCs using (5.3). Preliminary experiments with SNOPT all failed to find an HC, with many of the runs terminating at saddle points. Haythorpe developed a specialized interior point method for (5.3), which uses second derivatives and directions of negative curvature [43].

In this experiment, we compared ARCOPT with IPOPT on (5.3). Both solvers use second derivatives, but only ARCOPT makes explicit use of directions of negative curvature. During initial testing, we found that adding the constraint $P(x)^T e = e$ to (5.3) improved the likelihood of finding an HC with both solvers. We included the additional constraint for the results reported here.

5.2.1 10, 12, and 14 node cubic graphs

Cubic graphs are of interest because they represent difficult instances of HCP [42]. Variables corresponding to a node with only two edges may be fixed, because the cycle is forced to use each edge. Nodes in a cubic graph all have three edges and there is no possibility for reduction. Adding more edges only increases the likelihood of a Hamiltonian cycle. Cubic graphs are the most sparse and thus are the least likely to contain HCs in general.

Figure 5.1 shows a performance profile comparing ARCOPT and IPOPT on all 10, 12, and 14 node cubic graphs with HCs provided by Haythorpe [42]. Graphs without HCs were excluded. The test set includes a total of 571 graphs. In all cases both solvers were started at the analytic center of the feasible region. From this point, ARCOPT was able to find an HC in 79% (452) of the graphs while IPOPT solved 52% (298). This is indicated in Figure 5.1 by the maximum height of the plots corresponding to each solver. In general, ARCOPT required significantly fewer function evaluations as summarized in Table 5.2.

5.2.2 24, 30, and 38 node cubic graphs

ARCOPT and IPOPT were also tested on individual cubic graphs with 24, 30, and 38 nodes as well as a 30 node graph of degree 4. The graphs contained HCs and were provided by Haythorpe [42]. HCP becomes more difficult as the number of nodes is increased. Starting from the analytic center of the feasible region, ARCOPT found an HC in the 30 node graph of degree 4 and the 38 node cubic graph. IPOPT was unable to find an HC in all of these graphs when started from the same points.

To assess the relative likelihood of finding HCs, we performed an experiment using random starting points. For each graph, 20 random feasible starting points were generated in the following manner. First, a vector v was produced by sampling each element from a uniform distribution over $[0,1]$. The starting points were then selected by solving

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \|v-x\|_2^2 \\ \text{subject to} & Ax = e \\ & x \geq 0, \end{array}$$

where the linear constraints correspond to those for (5.3). Both solvers were started from the same points.

The results are summarized in Table 5.3 and Figure 5.2. Compared to IPOPT, ARCOPT was able to find HCs in larger graphs more reliably and efficiently.

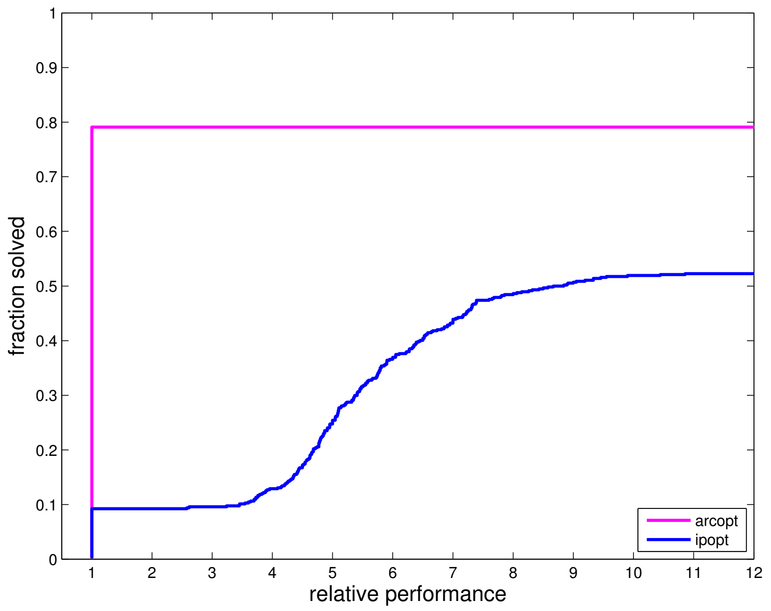

Figure 5.1: Performance profile on HCP comparing ARCOPT and IPOPT on all 10, 12, and 14 node cubic graphs with Hamiltonian cycles.

nodes	#	HC		ARCOPT	IPOPT
10	19	17	HCs found	15 (88.2%)	7 (41.2%)
			total function evaluations	275	1430
			average function evaluations	16.2	84.1
12	85	80	HCs found	67 (83.8%)	39 (48.8%)
			total function evaluations	1532	8120
			average function evaluations	19.1	101.5
14	509	474	HCs found	370 (78.1%)	252 (53.2%)
			total function evaluations	10804	53598
			average function evaluations	22.8	113.1
all	613	571	HCs found	452 (79.2%)	298 (52.2%)
			total function evaluations	12611	63148
			average function evaluations	22.1	110.6

Table 5.2: Summary of results comparing ARCOPT and IPOPT on all 10, 12, and 14 node cubic graphs with Hamiltonian cycles. The number of function evaluations reported includes the runs for which the solver failed to find a HC. The averages are reported over all runs. The # column reports the total number of cubic graphs. The HC column reports the total number of cubic graphs with HCs.

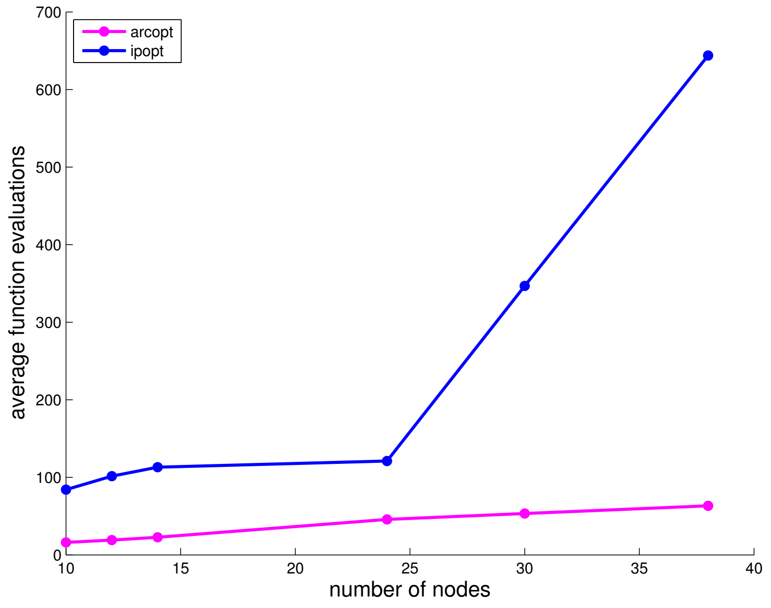

Figure 5.2: Average number of function evaluations to find HCs in cubic graphs of different sizes.

nodes	degree		ARCOPT	IPOPT
24	3	number solved	10 (50%)	3 (15%)
		total function evaluations	457	363
		average function evaluations	45.7	121
30	3	number solved	10 (50%)	3 (15%)
		total function evaluations	533	1040
		average function evaluations	53.3	346.7
38	3	number solved	10 (50%)	4 (20%)
		total function evaluations	633	2575
		average function evaluations	63.3	643.8
30	4	number solved	17 (85%)	8 (40%)
		total function evaluations	1573	1421
		average function evaluations	92.5	177.6

Table 5.3: Performance of ARCOPT and IPOPT on specific 24, 30, and 38 node cubic graphs and a 30 node graph of degree 4. For each graph the solvers were started from 20 randomly generated, feasible starting points. Function evaluations were reported for runs in which an HC was found.

5.3 The CUTEr test set

CUTEr stands for "Constrained and Unconstrained Test Environment revisited". It is a large set of optimization problems and associated tools created and maintained by Gould, Orban, and Toint [40]. This experiment compares ARCOPT, SNOPT, and IPOPT on:

- 127 unconstrained problems.
- 98 problems with bounds on the variables.
- 37 problems with a nonlinear objective and linear constraints.

There are many publicly available problems with linear constraints. However, the majority have linear or convex quadratic objective functions. This thesis is focused on problems with general nonlinear objective functions; thus we only test using the relatively small number of LC problems in CUTEr with nonlinear nonquadratic objective functions.

The experiments used function evaluations as the performance metric. Each solver was given a quota of 1000 iterations. The selected optimality and feasibility tolerances are shown in Table 5.4. Runs for which solution function values were not within 10^{-4} of the best solution were counted as failures. The number of function evaluations taken by each solver on each problem are shown in Tables A.1, A.2, and A.3 for unconstrained, bound constrained, and linearly constrained problems respectively.

The results are summarized by performance profiles in Figures 5.3, 5.4, and 5.5. Note that SNOPT does not use second derivatives and its under-performance was expected. IPOPT performed the best on unconstrained problems followed by ARCOPT then SNOPT. ARCOPT achieved best observed performance on 49% of the bound constrained problems followed by 37% for IPOPT and 14% for SNOPT. IPOPT was able to solve 81% of the bound constrained problems followed by 77% for SNOPT and 76% for ARCOPT. For linearly constrained problems, ARCOPT and IPOPT produced nearly equivalent profiles and performed better than SNOPT. Both IPOPT and SNOPT failed on 5 linearly constrained problems, while ARCOPT failed on 4.

solver	parameter	value
ARCOPT	`itermax`	1000
	`ptol` (primal tolerance)	1e-6
	`dtol` (dual tolerance)	1e-6
SNOPT	`major iterations limit`	1000
	`major optimality tolerance`	1e-6
	`major feasibility tolerance`	1e-6
IPOPT	`max_iter`	1000
	`tol`	1e-6
	`constr_viol_tol`	1e-6
	`compl_inf_tol`	1e-6

Table 5.4: Solver settings for CUTEr experiments.

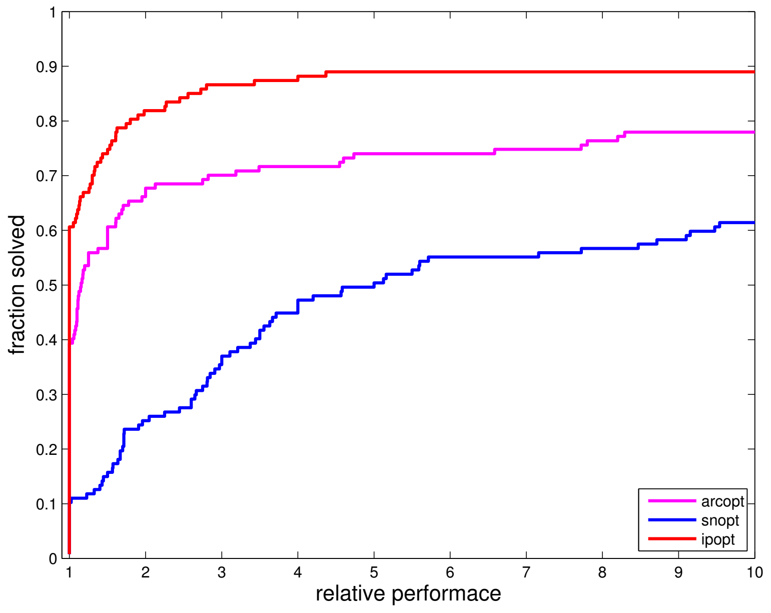

Figure 5.3: Performance profile on unconstrained problems.

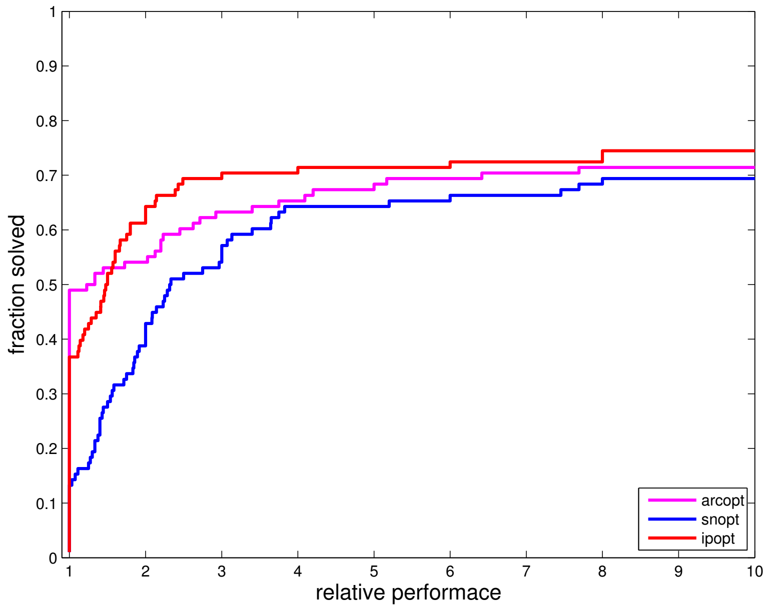

Figure 5.4: Performance profile on bound constrained problems.

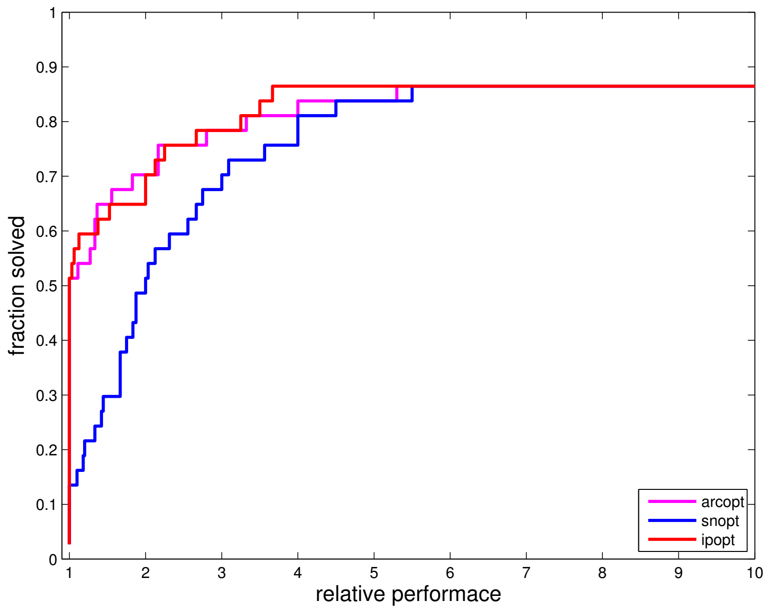

Figure 5.5: Performance profile on linearly constrained problems.

5.4 Quasi-Newton methods

Quasi-Newton methods emulate Newton's method by maintaining an approximation to the second derivative:

$$B_k \approx H_k.$$

The approximation is updated each iteration with a formula. Let $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. Two well known updates are

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

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{BFGS}$$

The updates are constructed to satisfy the *secant equation*,

$$B_{k+1} s_k = y_k, \tag{5.4}$$

which requires the updated approximation to match the most recent gradient evaluations.

The symmetric rank-1 update (SR1) is constructed by selecting a vector v and scalar σ such that $B_{k+1} = B_k + \sigma v v^T$ and (5.4) are satisfied. The problem in the context of line search is that B_{k+1} may not be positive definite even if B_k is. This presents the same problems as using an indefinite or singular Hessian when computing a descent direction. The SR1 update breaks down if the denominator is too small. Nocedal and Wright suggest that the update only be applied if

$$|s_k^T (y_k - B_k s_k)| \geq r \|s_k\| \|y_k - B_k s_k\|, \tag{5.5}$$

where $r \in (0, 1)$ with a suggested value of $r = 10^{-8}$ [53, p. 145].

The BFGS update is constructed such that B_{k+1} maintains positive definiteness. Vectors s_k and y_k must satisfy

$$s_k^T y_k > 0 \tag{5.6}$$

in order to keep $B_{k+1} \succ 0$. In the context of line search (5.6) is met if α_k is selected to satisfy the curvature condition (2.9) when $\phi_k''(0) \geq 0$. One issue with BFGS is that (5.6) may not be satisfied when a constraint is encountered. Second, the curvature condition (2.9) for general arcs may not directly imply (5.6), leading to degraded performance if the update is used. There are methods to handle the case where (5.6) is not met, e.g. the *damped* BFGS update [53, p. 537].

Maintaining positive definiteness for BFGS updates is not particularly difficult for linearly constrained problems. However, there are no assurances with updates for problems with nonlinear constraints. Although we do not address such problems here, it is of interest to investigate methods that generalize easily to nonlinearly constrained problems.

This experiment compares the methods: line search with BFGS updates (BFGS-LINE), arc search with BFGS updates (BFGS-ARC), and arc search with SR1 updates (SR1-ARC). NEM arcs as described in section 3.4 were used for arc search. All methods used the same code, which was adapted from ARCOPT. The optimality and feasibility tolerances were set to 10^{-6}. The iteration limit was set to 1000. SR1 updates were skipped if (5.5) was not satisfied. BFGS updates were skipped if arc or line search returned a step size that did not satisfy the curvature condition (2.9).

The results are summarized with performance profiles in Figures 5.6 and 5.7 for unconstrained and bound constrained problems respectively. Listings of function evaluation counts are shown in Tables A.4 and A.5. On unconstrained problems, SR1-ARC out-performed the other methods. On bound constrained problems, SR1-ARC and BFGS-LINE had similar performance initially, but BFGS-LINE was able to solve a greater total number of problems. BFGS-ARC had the worst performance on both test sets.

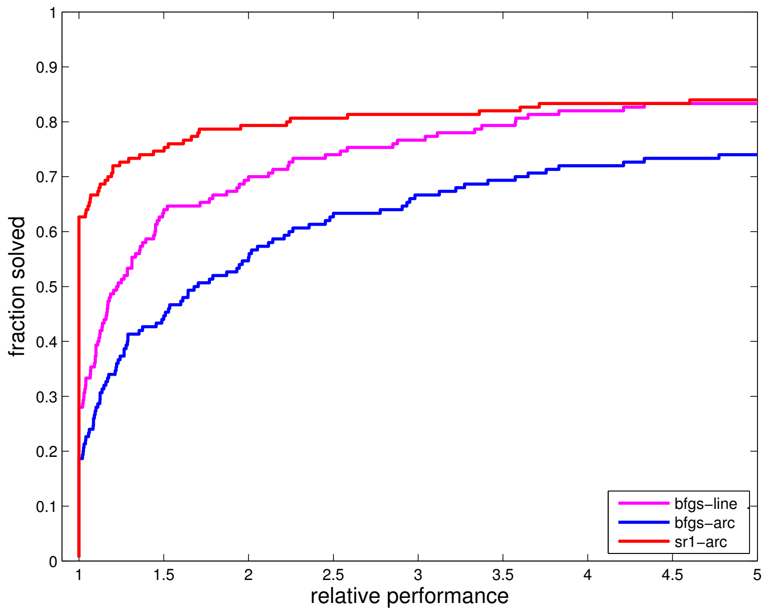

Figure 5.6: Performance profile for quasi-Newton experiments on unconstrained problems.

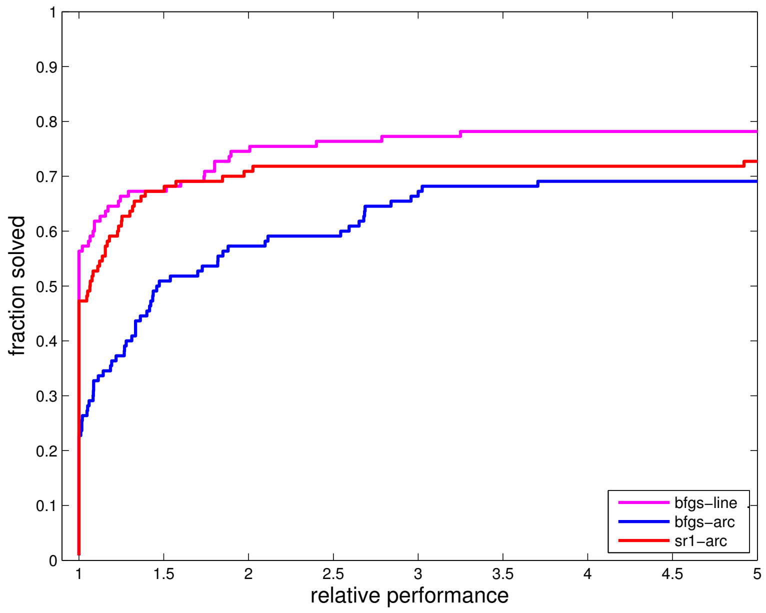

Figure 5.7: Performance profile for quasi-Newton experiments on bound constrained problems.

Chapter 6

Conclusions

6.1 Contributions

The first contribution of this thesis is the definition of a general arc search method for linearly constrained optimization. We show conditions under which convergence to second-order critical points is guaranteed. We also demonstrate the application of the convergence theory to a new arc we designate with NEM and several known methods: line search, curvilinear search, and modified gradient flow.

The second contribution is ARCOPT, an implementation of a reduced-gradient method using a NEM arc search for linearly constrained optimization. ARCOPT takes advantage of sparsity in the linear constraints and uses iterative methods for operations involving the reduced Hessian. These features allow ARCOPT to scale to problems with many variables. We document several practical considerations that arise in the construction of an arc search code. For example, the EXPAND procedure is a convenient way to remove undesirable roots in arc-constraint intersection equations when a constraint is deleted.

Numerical experiments with ARCOPT demonstrate good performance relative to SNOPT and IPOPT. ARCOPT outperforms IPOPT in both solution quality and efficiency on a continuous formulation of the Hamiltonian cycle problem. ARCOPT is competitive on a wide variety of problems in the CUTEr test set. We also show that the arc search framework allows for increased flexibility in solver development by comparing the BFGS and SR1 quasi-Newton updates in the same code.

6.2 Further work

The extension to nonlinear constraints could be done in a number of ways. The most direct approach would be to use an arc search method, such as ARCOPT, to solve the linearly constrained subproblem in a MINOS-type algorithm [30, 52]. A more challenging direction is to apply the ideas to an SQP method such as the one described by Murray and Prieto [50]. Note that Murray and Prieto's method uses a curvilinear search by necessity and treats linear and nonlinear constraints in the same manner.

Applying the ideas in this thesis would allow for the exploration of other arcs and separate treatment for linear constraints.

This thesis presents results of numerical experiments using dense quasi-Newton updates in an arc search method. The next step is to compare limited memory versions of the BFGS and SR1 updates, which can be applied to problems with a large number of variables. In nonlinearly constrained optimization, the Hessian of the Lagrangian need not be positive definite at a minimizer. Thus, maintaining a positive definite quasi-Newton approximation is a serious challenge. To avoid skipping updates, existing solvers employ various modifications, e.g. [34, Section 2.10] and [53, p. 536]. The SR1 update may be applied more often and without modification, because it has a less restrictive update requirement. The SR1 matrix may be indefinite and thus a better approximation to the Hessian of the Lagrangian.

Appendix A

Results tables

A.1 CUTEr results

Table A.1: Number of function evaluations taken to solve CUTEr **unconstrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
AKIVA	2	7	26	7
ALLINITU	4	11	18	14
ARGLINA	200	2	6	2
ARGLINB	200	*	*	3
ARGLINC	200	*	*	3
ARWHEAD	1000	7	24	6
BARD	3	11	24	8
BDQRTIC	1000	11	91	10
BEALE	2	10	17	19
BIGGS6	6	*	@	120
BOX	1000	8	84	13
BOX3	3	9	24	14
BRKMCC	2	4	11	4
BROWNAL	10	10	18	8
BROWNDEN	4	9	41	8
BRYBND	1000	20	45	16
CHAINWOO	1000	*	*	249
CHNROSNB	50	83	233	92
CLIFF	2	390	30	@
COSINE	1000	10	@	13
CRAGGLVY	500	15	122	14
CUBE	2	48	43	57

Table A.1: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
CURLY10	1000	22	*	22
CURLY20	1000	30	*	27
CURLY30	1000	@	*	30
DENSCHNA	2	7	12	7
DENSCHNB	2	7	11	24
DENSCHNC	2	11	21	11
DENSCHND	3	33	98	@
DENSCHNE	3	26	43	25
DENSCHNF	2	7	12	7
DIXMAANA	1500	10	27	8
DIXMAANB	1500	9	31	12
DIXMAANC	1500	16	32	9
DIXMAAND	1500	10	40	10
DIXMAANE	1500	13	166	11
DIXMAANF	1500	25	129	27
DIXMAANG	1500	29	139	18
DIXMAANH	1500	29	144	17
DIXMAANI	1500	199	891	24
DIXMAANJ	1500	164	377	20
DIXMAANK	15	15	124	13
DIXMAANL	1500	193	471	25
DIXON3DQ	1000	87	*	2
DJTL	2	*	*	*
DQDRTIC	500	3	28	2
EDENSCH	36	13	119	13
EG2	1000	39	@	5
EIGENALS	110	141	222	31
EIGENBLS	110	87	*	196
EIGENCLS	462	229	*	288
ENGVAL1	1000	*	36	9
ENGVAL2	3	23	36	33
ERRINROS	50	814	296	@
EXPFIT	2	10	22	9
EXTROSNB	1000	*	*	2334
FLETCBV2	1000	25	3	2
FLETCBV3	1000	*	*	*
FLETCHBV	1000	*	*	*
FLETCHCR	1000	*	*	*
FMINSRF2	961	171	*	275
FMINSURF	961	127	582	311
FREUROTH	1000	@	92	@
GENROSE	500	491	*	1256

Table A.1: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
GROWTHLS	3	843	182	178
HAIRY	2	63	55	109
HATFLDD	3	22	27	26
HATFLDE	3	23	43	21
HATFLDFL	3	165	470	*
HEART6LS	6	*	*	1433
HEART8LS	8	367	*	188
HELIX	3	11	32	25
HIELOW	3	9	33	9
HILBERTA	10	6	39	3
HILBERTB	50	3	11	2
HIMMELBG	2	5	13	14
HIMMELBH	2	6	10	24
HUMPS	2	302	179	488
JENSMP	2	11	42	10
KOWOSB	4	21	36	23
LIARWHD	1000	13	42	12
LOGHAIRY	2	598	372	*
MANCINO	30	6	13	5
MARATOSB	2	*	*	1804
MEXHAT	2	66	53	40
MEYER3	3	*	*	459
MODBEALE	200	8	@	9
MOREBV	1000	2	307	2
MSQRTALS	529	206	*	73
MSQRTBLS	529	164	*	47
NCB20B	1000	22	*	20
NONCVXU2	100	37	*	@
NONCVXUN	100	@	758	@
NONDIA	1000	23	111	5
NONDQUAR	1000	184	683	17
NONMSQRT	529	*	*	1002
OSBORNEA	5	*	115	37
OSBORNEB	11	17	95	24
OSCIPATH	500	5	28	5
PFIT1LS	3	*	460	704
PFIT2LS	3	315	148	202
PFIT3LS	3	959	301	344
PFIT4LS	3	*	484	549
POWELLSG	1000	18	*	17
POWER	1000	30	137	@
ROSENBR	2	25	49	45

Table A.1: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
S308	2	10	14	15
SCHMVETT	1000	4	66	4
SENSORS	100	@	75	@
SINEVAL	2	63	90	110
SINQUAD	1000	*	120	21
SISSER	2	15	25	15
SNAIL	2	92	133	148
SPARSINE	1000	975	*	14
SPARSQUR	1000	20	45	17
SPMSRTLS	1000	19	180	20
SROSENBR	1000	10	50	13
TESTQUAD	1000	4	*	2
TOINTGOR	50	8	160	8
TOINTGSS	1000	3	@	2
TOINTPSP	50	19	61	83
TOINTQOR	50	3	47	2
TQUARTIC	1000	2	97	2
TRIDIA	1000	3	403	2
VARDIM	200	457	*	28
WOODS	1000	553	305	84
ZANGWIL2	2	2	7	2
iter fails (*)		18	29	6
fval fails (@)		4	5	8
total fails		22	34	14

Table A.2: Number of function evaluations taken to solve CUTEr **bound constrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
3PK	30	580	447	12
ALLINIT	4	13	20	19
BDEXP	5000	@	75	@
BIGGSB1	100	200	*	14
BQP1VAR	1	2	4	6
BQPGABIM	50	12	23	18

Table A.2: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
BQPGASIM	50	12	25	18
CAMEL6	2	7	12	11
CHENHARK	100	38	*	13
CVXBQP1	100	100	@	@
DECONVB	61	30	@	@
EG1	3	@	@	8
EXPLIN	120	171	@	@
EXPLIN2	120	155	@	@
EXPQUAD	120	49	73	20
GRIDGENA	12482	5	*	8
HADAMALS	100	111	@	@
HART6	6	9	18	14
HATFLDC	25	5	26	6
HIMMELP1	2	12	19	12
HS1	2	32	50	53
HS110	200	@	*	*
HS2	2	7	16	17
HS25	3	@	@	43
HS3	2	4	9	5
HS38	4	57	119	77
HS3MOD	2	8	11	6
HS4	2	3	4	6
HS45	5	5	7	8
HS5	2	7	10	9
JNLBRNG1	529	100	100	13
JNLBRNG2	529	45	112	11
JNLBRNGA	529	496	82	11
JNLBRNGB	529	416	327	13
LINVERSE	1999	41	475	868
LOGROS	2	119	97	137
MCCORMCK	10000	7	116	8
MDHOLE	2	47	89	117
MINSURFO	5306	11	*	417
NCVXBQP1	100	104	@	@
NCVXBQP2	100	107	@	@
NCVXBQP3	100	@	20	@
NOBNDTOR	100	21	30	8
NONSCOMP	10000	10	131	@
OBSTCLAE	100	29	24	13
OBSTCLAL	100	38	21	14
OBSTCLBL	100	77	16	12
OBSTCLBM	100	50	14	10

Table A.2: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
OBSTCLBU	100	45	15	12
OSLBQP	8	2	6	12
PALMER1	4	31	32	1036
PALMER1A	6	*	99	92
PALMER1B	4	*	58	26
PALMER1E	8	@	245	@
PALMER2	4	14	43	2296
PALMER2A	6	*	117	205
PALMER2B	4	16	44	34
PALMER2E	8	*	247	22
PALMER3	4	30	@	412
PALMER3A	6	*	139	196
PALMER3B	4	20	47	15
PALMER3E	8	*	204	56
PALMER4	4	23	@	832
PALMER4A	6	*	101	119
PALMER4B	4	21	39	31
PALMER4E	8	*	176	46
PALMER5A	8	*	39	*
PALMER5B	9	*	*	5
PALMER5D	8	4	32	4
PALMER5E	8	*	*	*
PALMER6A	6	*	195	283
PALMER6E	8	*	139	60
PALMER7A	6	*	*	*
PALMER7E	8	*	*	*
PALMER8A	6	176	130	102
PALMER8E	8	*	89	30
PENTDI	5000	3	7	@
PROBPENL	500	31	6	6
PSPDOC	4	7	15	15
QR3DLS	155	231	*	114
QUDLIN	120	121	@	@
S368	100	@	28	@
SCOND1LS	502	*	*	2623
SIM2BQP	2	2	4	8
SIMBQP	2	4	8	8
SINEALI	1000	18	108	43
TORSION1	100	34	13	10
TORSION2	100	6	15	10
TORSION3	100	13	10	9
TORSION4	100	9	13	10

Table A.2: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems**. The number of variables is indicated by column n. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	ARCOPT	SNOPT	IPOPT
TORSION5	100	1	3	8
TORSION6	100	11	5	9
TORSIONA	100	42	14	10
TORSIONB	100	5	17	10
TORSIONC	100	17	11	8
TORSIOND	100	8	14	9
TORSIONE	100	1	3	8
TORSIONF	100	11	5	9
iter fails (*)		17	11	5
fval fails (@)		7	12	14
total fails		24	23	19

Table A.3: Number of function evaluations taken to solve CUTEr **linearly constrained problems**. The number of variables is indicated by column n. The number of linear constraints is indicated by column m. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	m	ARCOPT	SNOPT	IPOPT
DTOC1L	58	36	12	15	9
DTOC1L	598	396	12	24	9
EXPFITA	5	22	19	27	29
EXPFITB	5	102	32	32	33
EXPFITC	5	502	123	37	@
HAGER2	21	10	2	8	2
HAGER2	101	50	2	8	2
HAGER2	201	100	2	9	2
HAGER2	1001	500	2	11	2
HAGER4	21	10	8	15	11
HAGER4	101	50	28	12	10
HAGER4	201	100	53	11	10
HAGER4	1001	500	252	12	9
HIMMELBI	100	12	*	59	29
HIMMELBJ	45	14	*	195	*
HONG	4	1	14	13	9
HS105	8	1	@	@	25
HS112	10	3	39	33	18

Table A.3: (continued) Number of function evaluations taken to solve CUTEr **linearly constrained problems**. The number of variables is indicated by column n. The number of linear constraints is indicated by column m. Solver failures are indicated by *. Failures due to suboptimal objective function value are indicated by @.

Problem	n	m	ARCOPT	SNOPT	IPOPT
HS119	16	8	42	23	@
HS24	2	3	4	7	14
HS36	3	1	4	@	13
HS37	3	2	6	@	12
HS41	4	1	6	10	12
HS49	5	2	16	37	17
HS50	5	3	10	23	9
HS54	6	1	@	@	16
HS55	6	6	1	3	@
HS62	3	1	8	17	9
HS86	5	10	14	13	11
HS9	2	1	6	10	6
HUBFIT	2	1	4	11	9
LOADBAL	31	31	64	57	16
ODFITS	10	6	15	34	11
PENTAGON	6	15	8	15	17
QC	9	4	11	@	@
STANCMIN	3	2	3	6	11
TFI3	3	101	13	6	16
iter fails (*)			2	0	1
fval fails (@)			2	5	4
total fails			4	5	5

A.2 Quasi-Newton results

Table A.4: Number of function evaluations taken to solve CUTEr **unconstrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
AKIVA	2	14	23	19
ALLINITU	4	16	15	15
ARGLINA	200	3	3	3
ARGLINB	200	*	*	*

Table A.4: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
ARGLINC	200	*	*	*
ARWHEAD	100	*	*	10
BARD	3	24	26	24
BDQRTIC	100	*	*	66
BEALE	2	19	19	20
BIGGS6	6	44	40	40
BOX	100	18	33	14
BOX3	3	19	19	15
BRKMCC	2	7	7	7
BROWNAL	200	12	7	26
BROWNDEN	4	32	*	*
BROYDN7D	100	94	99	118
BRYBND	100	81	88	55
CHAINWOO	100	569	569	232
CHNROSNB	10	100	75	97
CLIFF	2	71	74	61
COSINE	100	38	47	28
CRAGGLVY	100	*	*	186
CUBE	2	40	47	68
CURLY10	100	254	846	248
CURLY20	100	227	617	207
CURLY30	100	211	497	179
DECONVU	61	60	58	441
DENSCHNA	2	12	12	11
DENSCHNB	2	9	9	9
DENSCHNC	2	23	23	21
DENSCHND	3	97	108	67
DENSCHNE	3	46	41	33
DENSCHNF	2	13	16	15
DIXMAANA	300	23	23	13
DIXMAANB	300	36	36	17
DIXMAANC	300	47	47	21
DIXMAAND	300	61	60	24
DIXMAANE	300	518	518	123
DIXMAANF	300	468	468	131
DIXMAANG	300	517	517	135
DIXMAANH	300	591	591	162
DIXMAANI	300	*	*	510
DIXMAANJ	300	*	*	827
DIXMAANK	15	114	114	57
DIXMAANL	300	*	*	1376
DIXON3DQ	100	173	165	152
DJTL	2	*	*	*

Table A.4: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
DQDRTIC	100	27	25	8
DQRTIC	100	302	268	208
EDENSCH	36	142	142	73
EG2	1000	5	5	5
EIGENALS	110	108	117	101
EIGENBLS	110	584	596	944
EIGENCLS	462	1413	1259	2801
ENGVAL1	100	68	68	38
ENGVAL2	3	36	36	40
ERRINROS	10	128	611	589
EXPFIT	2	19	17	19
EXTROSNB	100	*	*	*
FLETCBV2	100	203	2020	456
FLETCBV3	100	13	13	13
FLETCHBV	100	*	*	*
FLETCHCR	100	748	1468	928
FMINSRF2	121	146	235	143
FMINSURF	121	124	184	108
FREUROTH	100	27	27	24
GENROSE	100	428	637	446
GROWTHLS	3	2	2	2
HAIRY	2	32	27	26
HATFLDD	3	27	26	27
HATFLDE	3	36	35	42
HATFLDFL	3	911	225	149
HEART6LS	6	977	9694	*
HEART8LS	8	2055	1758	*
HELIX	3	29	29	31
HIELOW	3	*	*	18
HILBERTA	10	39	39	9
HILBERTB	50	11	11	8
HIMMELBB	2	19	13	13
HIMMELBF	4	48	820	124
HIMMELBG	2	18	18	14
HIMMELBH	2	11	11	9
HUMPS	2	236	204	196
HYDC20LS	99	*	*	*
INDEF	100	21	9	9
JENSMP	2	*	*	61
KOWOSB	4	24	27	27
LIARWHD	100	20	20	18
LOGHAIRY	2	10	22	96
MANCINO	30	30	30	14

Table A.4: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
MARATOSB	2	1533	4455	5152
MEXHAT	2	51	52	49
MEYER3	3	*	*	*
MODBEALE	200	*	288	*
MOREBV	100	203	2146	347
MSQRTALS	100	187	175	170
MSQRTBLS	100	189	187	177
NCB20	110	118	148	121
NCB20B	100	327	565	282
NONCVXU2	100	870	866	388
NONCVXUN	100	673	716	376
NONDIA	100	18	18	12
NONDQUAR	100	1683	1594	1419
NONMSQRT	100	667	*	*
OSBORNEA	5	83	4623	3748
OSBORNEB	11	76	78	81
OSCIPATH	100	45	55	36
PALMER1C	8	157	*	109
PALMER1D	7	118	*	33
PALMER2C	8	151	*	295
PALMER3C	8	141	*	49
PALMER4C	8	142	*	55
PALMER5C	6	30	29	9
PALMER6C	8	137	*	44
PALMER7C	8	143	*	47
PALMER8C	8	134	*	47
PENALTY1	100	82	89	96
PENALTY2	100	*	*	*
PENALTY3	200	*	*	*
PFIT1LS	3	738	65	2356
PFIT2LS	3	1494	3704	2244
PFIT3LS	3	1328	3918	7888
PFIT4LS	3	1608	5264	5792
POWELLSG	36	49	46	36
POWER	100	916	936	464
QUARTC	100	302	268	208
ROSENBR	2	41	42	59
S308	2	14	14	14
SCHMVETT	100	89	156	76
SENSORS	100	73	48	82
SINEVAL	2	95	111	158
SINQUAD	100	21	*	16
SISSER	2	25	25	22

Table A.4: (continued) Number of function evaluations taken to solve CUTEr **unconstrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
SNAIL	2	14	12	12
SPARSINE	100	303	303	134
SPARSQUR	100	108	109	71
SPMSRTLS	100	118	115	79
SROSENBR	100	15	19	18
TESTQUAD	1000	1357	2283	608
TOINTGOR	50	174	174	93
TOINTGSS	100	22	22	20
TOINTPSP	50	70	78	68
TOINTQOR	50	56	56	29
TQUARTIC	100	19	42	29
TRIDIA	100	103	99	105
VARDIM	100	33	33	33
WATSON	12	63	65	48
WATSON	31	75	220	1238
WOODS	100	42	39	32
ZANGWIL2	2	4	4	4
failures		18	28	14

Table A.5: Number of function evaluations taken to solve CUTEr **bound constrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
3PK	30	422	*	153
ALLINIT	4	19	18	20
ANTWERP	27	*	*	*
BDEXP	100	18	18	18
BIGGSB1	100	277	351	138
BLEACHNG	17	12	11	*
BQP1VAR	1	2	2	2
BQPGABIM	50	71	210	89
BQPGASIM	50	78	209	83
CAMEL6	2	13	13	12
CHARDIS0	200	3	3	3
CHEBYQAD	100	*	*	*
CHENHARK	100	237	378	125

Table A.5: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
CVXBQP1	100	108	107	101
DECONVB	61	197	*	243
EG1	3	12	11	11
EXPLIN	120	224	223	176
EXPLIN2	120	*	*	159
EXPQUAD	120	115	80	61
GRIDGENA	170	*	*	*
HARKERP2	100	334	377	311
HART6	6	22	40	22
HATFLDA	4	38	*	*
HATFLDB	4	36	*	*
HATFLDC	25	58	109	66
HIMMELP1	2	7	7	7
HS1	2	24	22	23
HS110	200	3	3	3
HS2	2	16	17	10
HS25	3	1	1	1
HS3	2	9	9	9
HS38	4	37	39	32
HS3MOD	2	9	7	5
HS4	2	3	3	3
HS45	5	4	4	5
HS5	2	12	17	13
JNLBRNG1	100	65	62	61
JNLBRNG2	100	41	39	35
JNLBRNGA	100	51	91	50
JNLBRNGB	100	51	86	41
KOEBHELB	3	211	*	2301
LINVERSE	199	*	*	*
LOGROS	2	114	168	225
MAXLIKA	8	100	284	118
MCCORMCK	100	13	13	14
MDHOLE	2	93	94	127
NCVXBQP1	100	103	103	101
NCVXBQP2	100	111	111	106
NCVXBQP3	100	117	115	106
NOBNDTOR	100	66	102	38
NONSCOMP	100	243	70	48
OBSTCLAE	100	62	69	48
OBSTCLAL	100	47	70	27
OBSTCLBL	100	53	51	43
OBSTCLBM	100	58	63	67
OBSTCLBM	100	58	63	67

Table A.5: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
OSLBQP	8	2	2	2
PALMER1	4	30	40	39
PALMER1A	6	150	2338	1287
PALMER1B	4	57	68	75
PALMER1E	8	238	*	*
PALMER2	4	22	38	27
PALMER2A	6	98	2268	9689
PALMER2B	4	39	41	41
PALMER2E	8	237	*	*
PALMER3	4	45	*	77
PALMER3A	6	126	3136	1276
PALMER3B	4	41	52	46
PALMER3E	8	196	*	*
PALMER4	4	39	60	79
PALMER4A	6	90	943	471
PALMER4B	4	*	47	47
PALMER4E	8	165	*	*
PALMER5A	8	*	*	*
PALMER5B	9	457	*	*
PALMER5D	8	26	24	8
PALMER5E	8	2170	*	4067
PALMER6A	6	190	1916	4636
PALMER6E	8	173	*	9223
PALMER7A	6	*	*	*
PALMER7E	8	2247	*	*
PALMER8A	6	166	446	221
PALMER8E	8	112	795	998
PENTDI	100	4	4	4
POWELLBC	200	*	*	*
PROBPENL	100	3	3	3
PSPDOC	4	14	16	14
QR3DLS	40	163	222	245
QRTQUAD	120	*	*	117
QUDLIN	120	188	188	147
S368	100	*	45	48
SCOND1LS	102	*	*	*
SIM2BQP	2	2	2	2
SIMBQP	2	4	4	4
SINEALI	100	61	62	*
SPECAN	9	50	61	33
TORSION1	100	39	84	14
TORSION2	100	54	48	56
TORSION3	100	6	8	6

Table A.5: (continued) Number of function evaluations taken to solve CUTEr **bound constrained problems** with quasi-Newton solvers. The number of variables is indicated by column n. Solver failures are indicated by *.

Problem	n	BFGS-LINE	BFGS-ARC	SR1-ARC
TORSION4	100	23	33	32
TORSION5	100	1	1	1
TORSION6	100	9	5	5
TORSIONA	100	35	74	55
TORSIONB	100	46	122	61
TORSIONC	100	6	8	6
TORSIOND	100	26	37	48
TORSIONE	100	1	1	1
TORSIONF	100	12	5	5
WEEDS	3	51	189	251
YFIT	3	87	1567	1293
failures		12	24	18

Bibliography

[1] Filippo Aluffi-Pentini, Valerio Parisi, and Francesco Zirilli. A differential-equations algorithm for nonlinear equations. *ACM Trans. Math. Softw.*, 10(3):299–316, August 1984.

[2] Alfred Auslender. Computing points that satisfy second order necessary optimality conditions for unconstrained minimization. *SIAM Journal on Optimization*, 20(4):1868–1884, 2010.

[3] William Behrman. *An efficient gradient flow method for unconstrained optimization.* PhD thesis, Stanford University, June 1998.

[4] Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 2nd edition, 1999.

[5] Paul T. Boggs. An algorithm, based on singular perturbation theory, for ill-conditioned minimization problems. *SIAM Journal on Numerical Analysis*, 14(5):830–843, 1977.

[6] C. A. Botsaris and D. H. Jacobson. A Newton-type curvilinear search method for optimization. *Journal of Mathematical Analysis and Applications*, 54(1):217–229, April 1976.

[7] Charalampos A. Botsaris. Differential gradient methods. *Journal of Mathematical Analysis and Applications*, 63(1):177 – 198, 1978.

[8] Charalampos A. Botsaris. An efficient curvilinear method for the minimization of a nonlinear function subject to linear inequality constraints. *Journal of Mathematical Analysis and Applications*, 71(2):482 – 515, 1979.

[9] Charalampos A. Botsaris. A Newton-type curvilinear search method for constrained optimization. *Journal of Mathematical Analysis and Applications*, 69(2):372 – 397, 1979.

[10] Mary A. Branch, Thomas F. Coleman, and Yuying Li. A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems. *SIAM Journal on Scientific Computing*, 21(1):1–23, 1999.

[11] A. A. Brown and M. C. Bartholomew-Biggs. Some effective methods for unconstrained optimization based on the solution of systems of ordinary differential equations. *Journal of Optimization Theory and Applications*, 62:211–224, 1989.

[12] Richard Byrd, Jorge Nocedal, and Richard Waltz. KNITRO: An integrated package for nonlinear optimization. In G. Pillo, M. Roma, and Panos Pardalos, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 35–59. Springer US, 2006.

[13] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[14] Richard H. Byrd, Robert B. Schnabel, and Gerald A. Shultz. Approximate solution of the trust region problem by minimization over two-dimensional subspaces. *Mathematical Programming*, 40:247–263, 1988.

[15] Andrew R. Conn, Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, 87:215–249, 2000.

[16] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust Region Methods*. Society for Industrial and Applied Mathematics, Philadephia, PA, 2000.

[17] Andrew R. Conn, Nick Gould, and Philippe L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. *Mathematical Programming*, 73:73–110, 1996.

[18] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS server. *Computational Science Engineering, IEEE*, 5(3):68–75, jul-sep 1998.

[19] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[20] Elizabeth D. Dolan, Jorge J. Moré, and Todd S. Munson. Optimality measures for performance profiles. *SIAM Journal on Optimization*, 16(3):891–909, 2006.

[21] Arne S. Drud. CONOPT: A large-scale GRG code. *ORSA Journal on Computing*, 6(2):207–216, 1994.

[22] Vladimir Ejov, Jerzy A. Filar, Walter Murray, and Giang T. Nguyen. Determinants and longest cycles of graphs. *SIAM Journal on Discrete Mathematics*, 22(3):1215–1225, 2008.

[23] Haw-ren Fang and Dianne OLeary. Modified Cholesky algorithms: a catalog with new approaches. *Mathematical Programming*, 115:319–349, 2008.

[24] M. C. Ferris, S. Lucid, and M. Roma. Nonmonotone curvilinear line search methods for unconstrained optimization. *Computational Optimization and Applications*, 6:117–136, 1996.

[25] Jerzy A. Filar, Michael Haythorpe, and Walter Murray. On the determinant and its derivatives of the rank-one corrected generator of a Markov chain on a graph. *Journal of Global Optimization*, pages 1–16, 2012.

[26] Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91:239–269, 2002.

[27] Anders Forsgren and Walter Murray. Newton methods for large-scale linear equality-constrained minimization. *SIAM Journal on Matrix Analysis and Applications*, 14(2):560–587, 1993.

[28] Anders Forsgren and Walter Murray. Newton methods for large-scale linear inequality-constrained minimization. *SIAM Journal on Optimization*, 7(1):162–176, 1997.

[29] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2nd edition, 2002.

[30] Michael P. Friedlander. *A Globally Convergent Linearly Constrained Lagrangian Method for Nonlinear Optimization*. PhD thesis, Stanford University, August 2002.

[31] Antonino Del Gatto. *A Subspace Method Based on a Differential Equation Approach to Solve Unconstrained Optimization Problems.* PhD thesis, Stanford University, June 2000.

[32] David Gay. A trust-region approach to linearly constrained optimization. In David Griffiths, editor, *Numerical Analysis*, volume 1066 of *Lecture Notes in Mathematics*, pages 72–105. Springer Berlin / Heidelberg, 1984.

[33] Philip E. Gill and Walter Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 7:311–350, 1974.

[34] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[35] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 8889(0):239 − 270, 1987.

[36] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45:437–474, 1989.

[37] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization.* Academic Press, 1982.

[38] Donald Goldfarb. Curvilinear path steplength algorithms for minimization which use directions of negative curvature. *Mathematical Programming*, 18:31–40, 1980.

[39] Nicholas I. M. Gould, Stefano Lucidi, Massimo Roma, and Philippe L. Toint. Exploiting negative curvature directions in linesearch methods for unconstrained optimization. *Optimization Methods & Software*, 14(1-2):75–98, 2000.

[40] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Softw.*, 29:373–394, December 2003.

[41] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[42] Michael Haythorpe. Cubic graph data, 2010. Personal communication.

[43] Michael Haythorpe. *Markov Chain Based Algorithms for the Hamiltonian Cycle Problem.* PhD thesis, University of South Australia, July 2010.

[44] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[45] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods.* SIAM, 1998.

[46] Chih-Jen Lin and Jorge J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[47] Garth P. McCormick. A modification of Armijo's step-size rule for negative curvature. *Mathematical Programming*, 13:111–115, 1977.

[48] Jorge J. Moré and Danny C. Sorensen. On the use of directions of negative curvature in a modified newton method. *Mathematical Programming*, 16:1–20, 1979.

[49] Jorge J. Moré and Danny C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

[50] Walter Murray and Francisco J. Prieto. A second derivative method for nonlinearly constained opimization. Technical report, Stanford University, Systems Optimization Laboratory, 1995.

[51] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[52] Bruce A. Murtagh and Michael A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. In *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, volume 16 of *Mathematical Programming Studies*, pages 84–117. Springer Berlin Heidelberg, 1982.

[53] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.

[54] Gerald A. Shultz, Robert B. Schnabel, and Richard H. Byrd. A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties. *SIAM Journal on Numerical Analysis*, 22(1):47–67, 1985.

[55] Danny C. Sorensen. Newton's method with a model trust region modification. *SIAM Journal on Numerical Analysis*, 19(2):409–426, 1982.

[56] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[57] Paul Tseng. Convergent infeasible interior-point trust-region methods for constrained minimization. *Siam Journal on Optimization*, 13(2):432–469, Oct 2002.

[58] Robert J. Vanderbei and David F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

[59] Jean-Philippe Vial and Israel Zang. Unconstrained optimization by approximation of the gradient path. *Mathematics of Operations Research*, 2(3):253–265, 1977.

[60] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

[61] Jianzhong Zhang and Chengxian Xu. A class of indefinite dogleg path methods for unconstrained minimization. *SIAM Journal on Optimization*, 9(3):646–667, 1999.