IPSOL: AN INTERIOR POINT SOLVER FOR NONCONVEX OPTIMIZATION PROBLEMS

A DISSERTATION
SUBMITTED TO THE PROGRAM IN SCIENTIFIC COMPUTING AND COMPUTATIONAL MATHEMATICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kaustuv

December 2008

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Walter Murray Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Michael Saunders

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mukund Thapa

Approved for the University Committee on Graduate Studies.

Abstract

For years, the Systems Optimization Laboratory (SOL) has been a center for research on large-scale nonlinear optimization. This research has given rise to reference solvers MINOS, NPSOL and SNOPT. In this thesis we present IPSOL, a prototype solver for general nonlinear optimization problem. IPSOL differs from the earlier (SOL) non-convex solvers in two ways. First, it uses second-derivative information, and second, it uses a barrier formulation to handle inequality constraints. Together, these features enable IPSOL to solve large-scale optimization problems efficiently.

IPSOL solves a sequence of equality-constrained log-barrier subproblems with decreasing values μ until convergence is attained. The subproblems are solved by an augmented Lagrangian SQP algorithm based on the theoretical framework developed by Murray and Prieto in [MP95]. A null-space method is used to solve each QP subproblem. This framework guarantees convergence to a second-order point when a direction of negative curvature for the reduced Hessian is used in conjunction with a descent direction. IPSOL uses the conjugate gradient algorithm to detect indefiniteness in the reduced Hessian and compute a direction of negative curvature. This direction is improved by application of a few steps of the Lanczos algorithm. A linesearch is then performed along the curve formed by a combination of the descent direction and the direction of negative curvature to get a steplength.

IPSOL has been prototyped in Matlab and tested on a subset of the CUTEr test problems. This thesis is focused on the algorithm and the details of implementation of IPSOL. We also discuss its performance on the CUTEr test set and compare the results against the current generation barrier solvers LOQO and IPOPT.

Acknowledgements

To all

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Basic Problem

The optimization problem of interest is

$$\boxed{\begin{array}{lrll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & & f(x) & \\ \text{s.t.} & & c_{\mathscr{E}}(x) & = b \\ & c_l \leq & c_{\mathscr{I}}(x) & \leq c_u \\ & x_l \leq & x & \leq x_u, \end{array}} \tag{1.1}$$

where $x \in \mathbb{R}^n$, $f \in C^2(\mathbb{R}^n)$, $f : \mathbb{R}^n \to \mathbb{R}$, $c_{\mathscr{E}} \in C^2(\mathbb{R}^n)$, $c_{\mathscr{E}} : \mathbb{R}^n \to \mathbb{R}^{m_{\mathscr{E}}}$, and $c_{\mathscr{I}} \in C^2(\mathbb{R}^n)$, $c_{\mathscr{I}} : \mathbb{R}^n \to \mathbb{R}^{m_{\mathscr{I}}}$. The problem addressed in this thesis is that of developing an algorithm (termed IPSOL) for finding a local minimizer of this class of problems. The main feature of this algorithm is that it can be used to solve large-scale problems efficiently. The scale of an optimization problem is defined in terms of the total number of variables and constraints. It is also tied to the available computational infrastructure. At the time of writing, a problem that has $n \geq 1000$ and $m_{\mathscr{E}} + m_{\mathscr{I}} \geq 1000$ is deemed large-scale. The efficiency of an optimization algorithm is measured in terms of the work required to compute the iterates or the number of function evaluations required to reach a sufficiently good estimate of a local minimizer from a given initial point.

Many theoretical algorithms with proofs of convergence have been proposed for solving (1.1). The proofs of convergence require that certain conditions hold at the solution. However, in practice, one or more of these assumptions may not be valid at the intermediate iterates or even at the solution. An implementation algorithm should follow the theoretical

algorithm when the underlying assumptions hold, but at the same time be robust enough to try and recover in case they breakdown. Moreover, an implementation needs to cope with the consequences of finite-precision arithmetic and the limitations on the linear algebraic operations that arise during the solution of large-scale problems. The challenge addressed in the development of IPSOL was to adhere closely to a theoretical algorithm while addressing such issues. IPSOL bridges the gap between the proof of convergence of a theoretical algorithm and its practical performance.

There are various mathematically equivalent formulations of (1.1). A very simple formulation is

$$\boxed{\begin{array}{lll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) & \\ \text{s.t.} & c(x) & \geq 0. \end{array}} \tag{1.2}$$

The formulation (1.2) can be obtained from (1.1) by a series of simple transformations and vice versa. In this thesis, formulation (1.1) is chosen over the simpler formulation (1.2) for ensuring computational efficiency of the algorithm.

The inequality constraints in (1.1) are relatively more difficult to handle than the equality constraints. An insight as to why this could be the case can be gained by looking at the first-order optimality conditions of (1.1) [NW06]. These equations are expressed in terms of the set of constraints active at a local optimizer. Choosing the subset of inequality constraints that are active at a solution is a combinatorial problem. This problem is computationally difficult when the total number of variables and inequality constraints is large.

In the next section, we describe a few approaches to solving optimization problems with inequality constraints. IPSOL uses a barrier approach for this class of optimization problem. In the subsequent section, we weigh the pros and cons of a barrier optimizer. Finally, we close the chapter by giving a brief history of barrier methods and a quick overview of optimizers using barrier approaches.

1.2 A Few Approaches to Solving Problems with Inequalities

As mentioned above, the presence of a large number of inequalities in an optimization problem may make it difficult to solve. There have been a wide variety of approaches proposed to solve this problem. For a comprehensive overview of these methods see [Mur74, Sto75, HM79, Tap80, SL99, GMS02]. Two classes of methods are generally viewed as the

most efficient:

- **Active-set approach:** These methods make a sequence of estimates of the correct active set. Most algorithms in this class solve the nonlinearly constrained problem (1.1) using a sequence of inequality-constrained quadratic programming (IQP) subproblems. The constraints of each IQP subproblem are linearizations of the constraints in the original problem, and the objective function of the subproblem is a quadratic approximation to the Lagrangian function. The solution of the IQP subproblem returns an independent set of constraints that are active at the IQP solution, which may be interpreted as an estimate of the active set of (1.1). It can be shown that the correct active set can be identified after a finite but potentially large number of iterations. In the absence of inequality constraints, the subproblems are equality-constrained quadratic programs (EQP). This approach of computing a solution of an optimization problem by solving a sequence of quadratic programs is called Sequential Quadratic Programming (SQP).

- **Barrier / Interior point approach:** Barrier methods were first introduced for inequality constrained problems of the form (1.2). They require a feasible initial point, and implicitly enforce the inequality constraints throughout. The inequality constraints are removed and the objective function is modified such that the search direction is biased to be directed away from the boundary. A sequence of such *unconstrained* optimization problems converging to the original problem is created and solved. Under suitable assumptions, it can be shown that the sequence of optimizers of these modified problems converges to a solution of (1.2). If equality constraints were present, they were removed by adding a penalty function to the objective. Again a sequence of *unconstrained* problems was solved. Current generation barrier algorithms use computationally efficient formulations of this basic idea. However, these algorithms converge to a solution by solving a sequence of nonlinear *equality-constrained* problems. Moreover, in these formulations, even on the problems of form (1.2), feasible iterates are no longer generated. So, in the strict sense, iterates are not in the interior of the feasible region. In the remaining part of the thesis the terms "barrier" and "interior point" would be used synonymously.

IPSOL uses a logarithmic barrier formulation to handle any inequalities in the problem. In the next section, we take a look at the pros and cons of barrier methods for large-scale optimization in comparison with traditional active-set approaches.

1.3 A Case for a Barrier Optimizer

As stated earlier, current algorithms for solving inequality-constrained optimization (1.1) fall in two categories: active-set methods and barrier methods. Active-set algorithms for nonlinear non-convex optimization are implemented in widely used general-purpose optimization codes such as MINOS [MS78, MS82], SNOPT [GMS02, GMS05] and filterSQP [FL02]. Barrier methods after their popularity in the 1960s went into decline, in part because of their inefficiencies and the availability of *new* algorithms to solve constrained optimization problems. Their re-emergence in the mid-1980s to solve LPs re-ignited interest in barrier methods. It was now seen that by no longer requiring the methods to solve a sequence of unconstrained problems, many of their deficiencies could be eliminated. Moreover, a clearer understanding of the ill-conditioned systems and how to circumvent them was achieved. Since then, barrier methods have made a deep foray into the world of convex optimization primarily because of the strong theoretical properties they offer [NN94] and their observed performance, which is far better than that predicted by the theory. Their adoption for nonconvex problems was slow because few of the theoretical results generalize to this case. Regardless of the loss of strong theoretical justifications, barrier methods have some attractive features:

+ An efficient active-set optimizer utilizes direct solvers to factorize the initial KKT system. Subsequent systems, often differing from the previous one by a rank-1 matrix, are solved by updating the factors of the previous system. Barrier methods need to solve relatively few linear systems of *fixed size* (compared to very many systems of smaller but varying size in case of active-set methods). The chosen solver can be iterative in nature, which automatically leads to a two-fold advantage:

  + Since all iterative solvers rely on matrix-vector multiplications, they are relatively easy to implement even in sparse cases.

  + Since the equations are of fixed size, their solution can be efficiently computed using a distributed linear system solver.

+ Barrier methods require the Hessian of the Lagrangian. The use of the Hessian enables convergence to points satisfying second-order optimality conditions. Methods based on an explicit Hessian are more efficient in terms of the number of function evaluations. Utilizing the explicit Hessian in barrier methods is a much simpler proposition than it is for active-set methods. This is partly because when the Hessian is indefinite it

is easy to see whether a modification is needed and also how to make a modification (see Chapter 3).

+ Active-set methods move from one working set to another and changes to the working set are very minimal. Typically, one or two constraints are added or deleted between successive working sets. In a large-scale case where there is a large number of working sets possible, an active-set solver may have to process many working sets before a correct active set is identified. An example to demonstrate this effect was first given by Klee and Minty for the simplex method in [KM72], where it is shown that an exponential number of active sets may be explored before an optimizer is found. Terlaky et al. [DNT08] have shown that barrier methods follow much the same path as simplex on the Klee-Minty examples. However, in practice, barrier algorithms converge to a sufficiently good estimate of the solution in very few (usually ≤ 40) iterations.

Barrier methods have some disadvantages:

– If the interior of the feasible region is empty, such as in mathematical programs with complementarity constraints (MPCC), interior point mthods would not be able to start [LPR96]. Fortunately, most problems arising out of physical sciences and engineering have a non-empty interior.

– The vanilla versions of barrier algorithms assume that the Hessian is easily computed, which might not be the case. Using an approximation to the Hessian can alleviate this problem to some extent.

– The name 'Interior Point' is a misnomer. It suggests that all the iterates generated are feasible, but most practical primal-dual implementations are in reality 'Infeasible Interior Point' algorithms. They only ensure that bounds on variables are always satisfied, while the constraints are satisfied only in the limit. Infeasible interior point approaches give a faster converging algorithm, especially in the presence of highly nonlinear constraints. The price paid is that if the algorithm is terminated prematurely, the final iterate might be infeasible, which might be of some concern to the practitioners[1].

[1]It should be noted that this particular trade-off is not limited to solvers based on interior point approaches but shows up in most present day NLP solvers. However if the infeasibility of the iterates is of prime concern then a solver based on Generalized Reduced Gradient (GRG) methods [LFR74], like CONOPT [Dru85], should be considered.

– The biggest drawback of an interior point solver is its inability to make use of a good initial starting point. The ideal starting point is equally away from all the boundaries or is on the central path[2]. Worse starting points are those not on the central path and close to a boundary of the inequality constraints. This is clearly a difficulty in many applications such as design optimization that model real-life scenarios. Experienced modelers often have a good idea of what the solution is going to look like. But this initial guess is likely to be off the central path and near a boundary, and hence a bad starting point.

– Closely related to the last difficulty is the issue of warm-starting a barrier method. Often in real-life applications, a sequence of closely related optimization problems are created and solved. All of these problems model the same physical phenomena but differ from one another in terms of the values taken by one or more parameters. Under mild assumptions it can be shown that the solution of an optimization problem is a continuous function of the parameters of the problem. Consequently, a small perturbation of the parameters leads to a small perturbation of the optimal solution. Thus it is desirable to initialize the new optimization problem (with perturbed parameters) using the optimal solution of the original one. However, some of the inequality constraints of the original problem are likely to be binding at its optimal solution. Consequently, this optimal point may also be close to a boundary of the inequality constraints of the perturbed optimization problem. This makes it a bad choice for initializing a barrier algorithm on the perturbed problem.

On the whole, if restart is not an issue, barrier methods have no major drawbacks, although their performance can be sensitive to the settings of various runtime parameters.

1.4 A Brief History of Barrier Methods

Perhaps no class of algorithms in the history of mathematical programming has a more fascinating past than the barrier methods [Wri05]. Also known as interior-point methods and potential methods, they first made their appearance in works of Frisch [Fri54, Fri55] and Caroll [Car61]. Fiacco and McCormick [FM90] developed a comprehensive theory about convergence properties of general barrier methods. This can be called the first era

[2]The central path for an optimization problem, say (1.6), is defined as the set of solutions for its log-barrier problem (1.7). Under some assumptions it can be shown that solutions of (1.7) are continuous functions of the parameter μ [FM90] and hence trace out a path as μ varies.

of research in barrier methods and is discussed in section 1.4.1. However, the popularity of these methods drastically dwindled in the years following the works of Lootsma [Loo69] and Murray [Mur69, Mur71], who showed that the Hessian of the barrier subproblems is ill-conditioned near the constraints.

The interest of the optimization community in Barrier methods was resurrected by Karmarkar's polynomial-time algorithm for linear programming [Kar84], which was formally shown to be equivalent to Newton's method applied to the log-barrier method [GMS$^+$86]. This renewed effort was mostly directed at the study of theoretical properties and the complexity of interior point methods for convex optimization [NN94] and the development of algorithms for the same [Meh92].

1.4.1 First Era: The Original Log-Barrier Formulation

The original log-barrier formulation applied the logarithmic barrier function directly to the inequality constraints. For example, in the absence of any equality constraints in (1.1), one would reformulate it as the following unconstrained problem:

$$\boxed{\begin{array}{ll}\underset{x\in\mathbb{R}^n}{\text{minimize}} & f(x) - \mu[e^T \ln(c_{\mathscr{I}}(x) - c_l) + e^T \ln(c_u - c_{\mathscr{I}}(x)) \\ & +e^T \ln(x - x_l) + e^T \ln(x_u - x)],\end{array}} \tag{1.3}$$

where e is a vector of 1s. (Terms involving infinite bounds are omitted.) Any equality constraints were accounted for by putting them as a penalty in the objective function. This reformulation of (1.1) as an unconstrained problem is similar to that proposed by Fiacco and McCormick [FM90]. In the 1960s, software for unconstrained problems was available and this approach enabled the state-of-the art unconstrained solvers to be applied to constrained problems.

Any solution of the above problem is feasible for (1.1). The basic idea is that one starts from an interior point of the feasible region of (1.1) and solves a sequence of subproblems (1.3) with values of $\mu_k \to 0$ to get a solution of x^* of (1.1).

It was shown by Lootsma [Loo69] and Murray [Mur69, Mur71] that the Hessian of the objective of (1.3) becomes increasingly ill-conditioned near the boundary of the feasible region and in particular in the neighborhood of the solution. Murray showed how the search direction could be computed accurately. However, ill-conditioning of the Hessian has other ramifications. For example, the quadratic rate of convergence of Newton's method is dependent on the Hessian being non-singular at the solution. Consequently, even the

theoretical algorithm has a poor rate of convergence. An additional difficulty in putting the logarithmic barrier directly on the constraints is that it requires an initial point that satisfies all the inequalities. This requirement may be stringent if the constraints are nonlinear.

1.4.2 Second Era: The Primal Log-Barrier Formulation

Both difficulties mentioned above are mitigated when the logarithmic barrier is applied only to bound constraints. This insight led to a formulation that is now known as the primal log-barrier formulation. We wish to point out two important aspects about this development:

- This formulation was first developed for *linear programs* even though the original log-barrier formulation works equally well for LP's.

- Although the application of Newton's method to this formulation is not marred by numerical ill-conditioning, its practical performance remains unsatisfactory.

To develop the primal log-barrier formulation of (1.1), all general inequality constraints in (1.1) are converted to equality constraints by the use of additional variables (called slack variables). With this transformation, (1.1) becomes

$$\boxed{\begin{array}{lrl} \underset{x,s}{\text{minimize}} & f(x) & \\ \text{s.t.} & c_{\mathscr{E}}(x) - b & = 0 \\ & c_{\mathscr{I}}(x) - s & = 0 \\ & x_l \leq x & \leq x_u \\ & c_l \leq s & \leq c_u. \end{array}} \tag{1.4}$$

With $\overline{x} = [x; s]$, $l = [x_l; c_l]$, $u = [x_u; c_u]$ and $c(\overline{x}) = [c_{\mathscr{E}}(x) - b; c_{\mathscr{I}}(x) - s]$ the equations become

$$\boxed{\begin{array}{lrl} \underset{\overline{x}}{\text{minimize}} & f(\overline{x}) & \\ \text{s.t.} & c(\overline{x}) & = 0 \\ & l \leq \overline{x} & \leq u. \end{array}} \tag{1.5}$$

We introduce additional variables $t_l \in \mathbb{R}^{n+m_{\mathscr{I}}}$ and $t_u \in \mathbb{R}^{n+m_{\mathscr{I}}}$ that convert the bound constraints into positivity constraints:

$$\begin{array}{lrcl} \underset{\overline{x},t_l,t_u}{\text{minimize}} & f(\overline{x}) & & \\ \text{s.t.} & c(\overline{x}) & = & 0 \\ & \overline{x} - t_l & = & l \\ & -\overline{x} - t_u & = & -u \\ & t_l, t_u & \geq & 0. \end{array} \tag{1.6}$$

Now introducing the barrier on the t's we get the following barrier subproblem:

$$\begin{array}{lrcl} \underset{\overline{x},t_l,t_u}{\text{minimize}} & f(\overline{x}) - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u)] & & \\ \text{s.t.} & c(\overline{x}) & = & 0 \\ & \overline{x} - t_l & = & l \\ & -\overline{x} - t_u & = & -u. \end{array} \tag{1.7}$$

where e_l and e_u are vectors of 1s. In the primal log-barrier approach, the above equality-constrained problem is solved with decreasing values of μ to get a solution of (1.1). Application of Newton's method to the optimality conditions of (1.7) restricts the elements causing ill-conditioning to be on the diagonal of the Hessian. Because of this special distribution of elements causing ill-conditioning, it can be shown that the ill-conditioning is "benign" and the Newton search directions can be computed accurately.

This reformulation does remove the effects of ill-conditioning in the original formulation. However, the application of Newton's method to the optimality conditions of (1.7) is still marred by very slow convergence. The successive iterates of Newton's method are constructed by solving a quadratic approximation to (1.7) at the current iterate. Near any active constraint, the logarithmic barrier terms are not well approximated by a quadratic polynomial, leading to slow and inefficient progress of Newton's method.

1.4.3 The Primal-Dual Equations

The extensive research that followed the re-discovery of barrier algorithms led to the discovery of a simple alternative reformulation of the optimality equations for (1.7) [Meg89]. Application of Newton's method to the resulting equations is much more efficient. For the special case of linear programming, there are as many as six reformulations proposed in [GMPS95]. For the nonlinear non-convex case, the computationally efficient reformulation is called primal-dual equations. These equations are developed in chapter 2. Their

linearization is solved to compute a search direction at an iterate.

1.5 Similar Projects

Currently, several projects use barrier methods to solve general nonlinear non-convex programs. In Table 1.5 we give highlights of these projects. The published performances of IPOPT and LOQO are used to benchmark the algorithm proposed in this thesis.

Solvers	Algorithm	Released
IPOPT [WB06]	Primal-dual system with filter based linesearch	2000
KNITRO [BNW06]	Trust-region	2001
LOQO [Van99]	Primal-dual system with merit function based linesearch	1999

Table 1.1: Current interior point solvers

1.6 Outline of the Remaining Chapters

The basic equations that are solved to get the search directions are developed in chapter 2. Additionally, chapter 2 describes the strategy used to initialize the solver. Chapter 3 is about the computational algorithm used to obtain the search directions from the equations developed in chapter 2. Chapter 4 describes the Augmented Lagrangian based linesearch algorithm for selecting a step along the computed search directions. Chapter 5 discusses the strategy of reducing μ from one subproblem to another. It also notes various termination criteria for the subproblem and the overall problem. Chapter 5 also contains the computational results of IPSOL on a subset of the CUTEr test set.

Chapter 2

Development of Equations and Initialization

2.1 Introduction

In this chapter, we present an outline of IPSOL. We derive the set of equations that are solved to get the search direction for the primal and dual variables. Then, a broad overview of the entire algorithm is presented. Finally, the strategy used for initializing the solver and the presolve phase are described. (The term "search direction" as used in the thesis refers to the direction of descent in both primal and dual variables as well as to the direction of negative curvature, provided one exists.)

2.2 Log-Barrier Problem Revisited

A barrier algorithm like IPSOL transforms the given optimization problem (1.1) into an equality-constrained problem (1.7). This transformed problem is reproduced here for the sake of completeness:

$$\begin{array}{lrcl} \underset{\overline{x},t_l,t_u}{\text{minimize}} & f(\overline{x}) & - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u)] \\ \text{s.t.} & c(\overline{x}) & = & 0 \\ & \overline{x} - t_l & = & l \\ & -\overline{x} - t_u & = & -u. \end{array} \tag{2.1}$$

In IPSOL, the search directions are obtained by applying Newton's method to the optimality conditions associated with this problem.

2.2.1 One Sided Bounds and Free Variables

In (2.1), it is implicitly assumed that all the general inequality constraints are upper as well as lower bounded. A similar assumption is made regarding the bounds on the decision variables. In formulation (2.1), a one-sided bound (on the general constraints or on the decision variables) is represented using $\pm\infty$ as the other bound. However, the inclusion of ∞ in the problem causes serious degradation in the performance of most algorithms, IPSOL being no exception. A more computationally efficient log-barrier formulation that explicitly includes the cases of one-sided bounds and free variables is:

$$\begin{array}{lrcll} \underset{\overline{x},t_l,t_u}{\text{minimize}} & f(\overline{x}) - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u)] & & & \\ \text{s.t.} & c(\overline{x}) & = & 0 & : y \\ & E_l\overline{x} - t_l & = & l & : z_l \\ & -E_u\overline{x} - t_u & = & -u & : z_u. \end{array} \tag{2.2}$$

In this formulation E_l, E_u are matrices that extract a sub-vector out of a given vector and y, z_l, z_u are the corresponding vectors of Lagrange multipliers of appropriate dimensions. The bounds $t_l > 0$ and $t_u > 0$ are present implicitly. The Lagrangian of this equality-constrained problem is defined as:

$$\begin{array}{rl} \mathscr{L}(\overline{x}, t_l, t_u, y, z_l, z_u) := & f(\overline{x}) - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u)] \\ & -y^T c(\overline{x}) - z_l^T (E_l\overline{x} - t_l - l) - z_u^T (-E_u\overline{x} - t_u + u). \end{array} \tag{2.3}$$

2.3 The Optimality Equations and Their Linearization

The optimality equations for (2.2) are

$$\begin{array}{rrrrcrl} -\nabla f(\overline{x}) & +\nabla c(\overline{x})\, y & +E_l^T z_l & -E_u^T z_u & = & 0 & (L) \\ & & T_l z_l & & = & \mu e_l & (pcl) \\ & & & T_u z_u & = & \mu e_u & (pcu) \\ c(\overline{x}) & & & & = & 0 & (cons) \\ E_l\overline{x} & & -t_l & & = & l & (low) \\ -E_u\overline{x} & & & -t_u & = & -u & (upp), \end{array} \tag{2.4}$$

where $T_{l,u} = \mathrm{diag}(t_{l,u})$ and $e_{l,u}$ are unit vectors of appropriate dimensions. Again, the conditions $t_l > 0$ and $t_u > 0$ are implicitly assumed and together with (pcl) and (pcu) ensure that the logarithms are well defined[1].

Linearization of the optimality conditions (2.4) at a point $(\overline{x}, t_l, t_u, y, z_l, z_u)$ gives rise to the following Newton equations:

$$\begin{bmatrix} -\nabla^2\mathscr{L} & & & \nabla c(\overline{x}) & E_l^T & -E_u^T \\ & Z_l & & & T_l & \\ & & Z_u & & & T_u \\ \nabla c(\overline{x})^T & & & & & \\ E_l & -I & & & & \\ -E_u & & -I & & & \end{bmatrix} \begin{bmatrix} \delta\overline{x} \\ \delta t_l \\ \delta t_u \\ \delta y \\ \delta z_l \\ \delta z_u \end{bmatrix} = \begin{bmatrix} r_L \\ r_{pcl} \\ r_{pcu} \\ r_{cons} \\ r_l \\ r_u \end{bmatrix}. \tag{2.5}$$

where $Z_{l,u} = \mathrm{diag}(z_{l,u})$ and r_L, r_{pcl}, r_{pcu}, r_{cons}, r_l, r_u are residuals at the current iterate of the equations (L), (pcl), (pcu), (cons), (l), (u) respectively. This large and unsymmetric system, after elimination of $\delta z_l, \delta z_u, \delta t_l, \delta t_u$ (in that order), is reduced to a much smaller symmetric system[2]

$$\begin{bmatrix} -H_{pd} & A^T \\ A & \end{bmatrix} \begin{bmatrix} \delta\overline{x} \\ \delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}, \tag{2.6}$$

where

$$\begin{aligned} H_{pd} &= \nabla^2\mathscr{L} + E_l^T T_l^{-1} Z_l E_l + E_u^T T_u^{-1} Z_u E_u \\ r_1 &= r_L - E_l^T T_l^{-1}(r_{pcl} + Z_l r_l) + E_u^T T_u^{-1}(r_{pcu} + Z_u r_u) \\ r_2 &= r_{cons} \\ A &= \nabla c(\overline{x})^T. \end{aligned} \tag{2.7}$$

Equations (2.6) are referred to as the primal-dual system. The matrix H_{pd} is called the primal-dual Hessian. It is composed of two parts, the regular Hessian of the Lagrangian, $\nabla^2\mathscr{L}$, and the diagonal matrix $E_l^T T_l^{-1} Z_l E_l + E_u^T T_u^{-1} Z_u E_u$. This structure is of utmost importance in the numerical computation of the search directions. This subtlety shall be elaborated in chapter 3. In order to reduce the notational clutter, the following simpler

[1] The optimality equations (2.4) offer another view to motivate the barrier approach. With $\mu = 0$, this system of equation reduces to the optimality condition of the formulation (1.6). So, in effect a barrier approach solves the optimality condition of the optimization problem after relaxing the complementarity equations.

[2] The primary purpose of this reduction is to gain a computational edge. It is well known in the linear algebra world that large unsymmetric systems are typically more difficult to solve than smaller symmetric systems.

representation of (2.6) shall be used:

$$\begin{bmatrix} -(H+D) & A^T \\ A & \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}, \tag{2.8}$$

where $H = \nabla^2 \mathscr{L}$ and $D = E_l^T T_l^{-1} Z_l E_l + E_u^T T_u^{-1} Z_u E_u$. The algorithm to compute the search directions using (2.8) is described in detail in chapter 3.

2.4 Overview of the Algorithm

IPSOL solves (2.2) with a sequence of decreasing values of μ until convergence is attained. An overview of the algorithm is presented as Algorithm 1. The basic structure of the algorithm involves *major* and *minor* iterations. At each major iteration, the value of μ is held constant and the subproblem (2.2) is solved. Solving such a subproblem is itself an iterative procedure, with the minor iterations of IPSOL being the iterations of the algorithm used for solving (2.2). After each subproblem converges, and if the convergence criteria for the overall problem is not satisfied, then the value of μ is decreased and another major iteration is performed.

An equality-constrained program like (2.2) can be solved using a wide variety of approaches such as the Bound Constrained Lagrangian (BCL) approach [CGT92], Linearly Constrained Lagrangian (LCL) approach [Rob72, FS05], Sequential Quadratic Programming (SQP) [BT95], and the Generalized Reduced Gradient (GRG) approach [LFR74].

It is a widely accepted view amongst the present day optimization community that out of these approaches, the SQP methods are most suited for a general large-scale nonlinear optimization problem. SQP algorithms are very robust, especially in the presence of highly nonlinear objective and constraints. SQP methods that use second-derivative information generally require fewer function and gradient evaluations (than methods using just the first derivatives), making them the method of choice for solving the subproblem (2.2). The particular SQP method that is used in IPSOL to solve (2.2) is described in [MP95]. Murray and Prieto's algorithm is a linesearch based second-derivative SQP method that uses an augmented Lagrangian merit function to ensure global convergence. The algorithm is designed for a problem with inequality constraints *but can be used equally efficiently for an equality-constrained problem like (2.1).* The algorithm requires only an approximate solution to the SQP subproblems. This feature of the algorithm is crucial for IPSOL, because at any minor iteration IPSOL uses an iterative solver to compute a good approximation to the

begin
Initialize $\mu_o, \overline{x}_0, y_0, t_{l_0}, t_{u_0}, z_{l_0}, z_{u_0}$
[Section 2.5]

[Major Iterations]
while *original problem is not converged [Section 5.3]* **do**

Solve equality-constrained subproblem (2.2)

[Minor Iterations]
while *subproblem is not converged [Section 5.3]* **do**
Compute search directions using (2.6)
[Chapter 3]

Compute a step length to achieve a sufficient decrease in the augmented Lagrangian merit function
[Chapter 4]

Update variables
[Chapter 4]
end

Initialize the new subproblem using the optimal value.
$\overline{x}_k \leftarrow x^*, t_{l_k} \leftarrow t_l^*, t_{u_k} \leftarrow t_u^*$
$y_k \leftarrow y^*, z_{l_k} \leftarrow z_l^*, z_{u_k} \leftarrow z_u^*$
$k \leftarrow k + 1$

Decrease μ
[Section 5.2]
end
end

Algorithm 1: Overview of IPSOL

search direction.

2.5 Selection of an Initial Point

Nonconvex nonlinear programs (NCP) are generally characterized by the presence of multiple local minima. An NCP algorithm converges to a particular minimizer that, out of the many possible ones, is greatly dictated by the choice of the initial point.

A complete description of a starting point requires specifying an initial value of the barrier parameter μ_0 and primal, dual, and slack variables ($[\overline{x}; y; z_l; z_u; t_l; t_u]$). In an optimization problem arising from a real-life application, primal variables often have a physical or economic relevance. Many times, based on prior experience with the model, the user is able to specify a good initial value for the primal variables. However, other variables that lack any obvious physical or economic interpretation are usually not provided by the user. For barrier algorithms, the ideal initial points are ones that lie on the central path.

In IPSOL, $\overline{x}$ is initialized to the user-provided value $\overline{x}_0$. In the absence of any input from the user, $\overline{x}$ is initialized to a zero vector. A conservative initial value for μ is to set it to a large number such as 10^3 to ensure that the initial search directions are strongly biased away from the boundaries. However, this choice is computationally inefficient as the initial iterations are not focused on reducing the value of the objective function. Additionally, it may require a large number of major iterations to drive down μ to 0. A more aggressive choice would be to set the initial μ close to 0 (such as 10^{-3}). In this case, there is a danger that the iterates prematurely converge to the boundaries and get stuck there. In IPSOL, μ is initialized to be a multiple of $\|\nabla f(\overline{x}_0)\|_\infty$. A steepest descent step in this case is dominated by logarithmic repulsion, which keeps the iterates away from the boundaries. To prevent the initial μ from getting too small or too large, we restrict it to be between a lower and upper bound. In short, μ is initialized as

$$\mu_0 \;=\; \max(\kappa_1, \min(\kappa_2 \|\nabla f(\overline{x}_0)\|_\infty, \kappa_3)). \tag{2.9}$$

The default value for κ_1 is 0.1, and 10 for both κ_2 and κ_3, so $\mu_0 \in [0.1, 10]$.

The slack variables are initialized as

$$\begin{aligned} t_{l_0} &= \max(E_l \overline{x}_0 - l, \sqrt{\mu_0}) \\ t_{u_0} &= \max(u - E_u \overline{x}_0, \sqrt{\mu_0}). \end{aligned} \tag{2.10}$$

This choice will satisfy the bound constraint on a component of $\overline{x}$ iff the particular component is at least $\sqrt{\mu_0}$ distance inside the bound. The possible failure of t_{l_0} or t_{u_0} to satisfy (low) or (upp) in (2.4) is not of concern as these constraints get satisfied at any iteration in which a unit step is taken. Furthermore, once these constraints are satisfied and search directions are computed accurately, they continue to remain satisfied at all later iterates. The dual variable z's are all initialized to $\sqrt{\mu_0}$. Finally, y is initialized to the least squares solution of the equation

$$\nabla c(\overline{x}_0)\ y \quad = \quad \nabla f(\overline{x}_0) - E_l^T z_{l_0} + E_u^T z_{u_0}. \tag{2.11}$$

2.6 Presolving

Presolving is a set of techniques applied to linear and bound constraints in the hope of simplifying the problem being solved. For linear and quadratic programs, a wide variety of techniques have been developed to reduce the number of variables and constraints as well as to detect some kinds of infeasibility [Gon97, GT04]. For a general nonlinear program, we just aim at reducing the number of redundancies in the problem, yielding an easier program to solve.

2.6.1 Fixed Variables

Variables whose values are fixed are called fixed variables. Variables may be fixed by the user to see the behavior of the model w.r.t to other variables. In optimal control problems, variables are fixed to specify the initial or final state of the system. The presence of fixed variables results in an empty interior, and may cause the interior point algorithms to fail. IPSOL solves the problem in the subspace of the feasible region defined by treating the fixed variables as constants.

2.6.2 Redundant Constraints

A constraint is redundant if its lower bound is $-\infty$ and upper bound is ∞. These bounds may be set by the user to see the performance of the model in the absence of these constraints. The presence of $\pm\infty$ in the bound vectors l, u makes the problem numerically difficult to solve. However, the redundant constraints do not influence either the feasible region or the solution set of the problem. So dropping them out of the problem definition,

as is internally done by IPSOL, is safe as it does not alter either the feasible region or the solution set.

2.7 Conclusion

In this chapter, an overview of IPSOL was presented. The basic equations, which form the core of the computation of the search direction, were derived. The choice used for initializing the solver and the pre-solve phase was also described. In the next two chapters, we describe the computation of the search direction and the linesearch algorithm. These two are the critical building blocks for solving the subproblems (2.2).

Chapter 3

Search Direction Computations

3.1 Introduction

An approximate solution of the barrier subproblem (2.1) for a specific choice of μ is obtained by generating a sequence of approximations. The elements of this sequence are obtained by approximately solving the primal-dual equations (2.8), which for reference are noted here:

$$\begin{bmatrix} -(H+D) & A^T \\ A & \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} . \tag{3.1}$$

These equations may require modification. We show later how to detect the need for a modification and what modifications to make. The existence and uniqueness of the solution of (3.1) are guaranteed under a few assumptions on the matrices involved.

Theorem 3.1. *Let Z be a basis for the null space for the constraint matrix A. If A has full row rank and the reduced Hessian $Z^T(H+D)Z$ is positive definite then the solution of the modified primal-dual system (3.1) exists and is unique.*

Proof. The proof can be found in textbooks such as [NW06]. □

The presence of rank deficiency in A can cause the above system to be inconsistent, while indefiniteness in the reduced Hessian can lead to search directions seeking a maximizer or a saddle point rather than a minimizer.

In the neighborhood of a non-degenerate solution, the assumptions in the above theorem are satisfied. However, this might not be the case outside this region. The challenge in solving the modified primal-dual system is to detect a possible rank deficiency and/or

indefiniteness and to modify the system, in case it is detected. Further, in order to permit a superlinear rate of convergence, the systems should not be modified in the neighborhood of a solution.

The aim of the following sections is to explain the detection/modification algorithm that has been implemented in IPSOL. In order to avoid conceptual clutter, the exposition proceeds by analyzing the algorithm on a sequence of increasingly difficult cases.

Section 3.2 discusses the need for scaling (3.1) and various scaling strategies used in IPSOL. Section 3.3 is about solving the scaled modified primal-dual system by the reduced Hessian approach. In particular, there is detailed discussion for efficient computation of a null-space basis for the constraint matrix A. Section 3.4 details the regularizations and strategies used in case indefiniteness is detected in the reduced Hessian. Finally, section 3.5 is about the detection of rank deficiency in A and the regularizations used overcome it.

3.2 Scaling the Modified Primal-Dual System

Irrespective of the linear algebraic properties of the matrices H and A, (3.1) has elements of widely varying magnitude. This primarily comes from the entries of the diagonal matrix D. As $\mu \to 0$, D has some elements $\mathrm{O}(\mu)$ and others $\mathrm{O}\left(\frac{1}{\mu}\right)$. Poor scaling of the problem can also arise within the matrices H and/or A, preventing accurate computation of $(\delta x, \delta y)$.

To reduce the consequences of poor scaling, the matrices H, D and A are scaled in an attempt to make all non-zero elements of the primal-dual matrix of similar magnitude (ideally $\mathrm{O}(1)$). Although H, D, and A change every *minor* iteration, we rescale them every *major* iteration. The scaling strategy used in IPSOL is a modification of the classical p-norm scaling, with the modification ensuring that the scaling matrices are symmetric. Algorithm 2 is the pseudo-code representation of this scaling procedure.

Clearly, the p-norm scaling requires that individual elements of the matrices be available for the computation of various norms. This requirement is easily satisfied for matrices A and D, that are assumed to be explicitly given. However, there may be cases in which H is available only as an operator for multiplication with a vector. It should be noted that the norm-based scaling algorithms require a ballpark estimate of the p-norms. The algorithm of Chen and Demmel [CD00] can be used to estimate the 2-norms of the rows of the H. This procedure is stated in Algorithm 3. IPSOL offers scaling in the 1, 2 and inf norms. The default option is to scale the system in the 1-norm. *At this point we would like to point out that IPSOL does not require an explicit representation of H, either for this or for*

any other computation. Any implicit representation which allows for efficient matrix-vector multiplications is acceptable.

Algorithm: scaleprimal-dual

Input: $H \in \Re^{n\times n}$, $D \in \Re^{n\times n}$, $A \in \Re^{m\times n}$
Output: Diagonal matrices $S_1 \in \Re^{n\times n}_{++}$ and $S_2 \in \Re^{m\times m}_{++}$

begin

Initialization.
$S_1 = I_{n\times n}$
$S_2 = I_{m\times m}$

Iteratively computing the scalings.
for $t = 1$ **to** *iterMax* **do**

Scale the matrices using current scaling factors.
$H = S_1 H S_1$
$D = S_1 D S_1$
$A = S_2 A S_1$

Compute scalings for the current matrices.
for $i = 1$ **to** n **do** $\sigma_1(i) = \|(\|H(i,:)\|_p, D(i,i), \|A(:,i)\|_p)\|_p$
for $i = 1$ **to** m **do** $\sigma_2(i) = \|A(i,:)\|_p$

Update the scaling matrices.
for $i = 1$ **to** n **do** **if** $\sigma_1(i) \neq 0$ **then** $S_1(i,i) = S_1(i,i)/\sqrt{\sigma_1(i)}$
for $i = 1$ **to** m **do** **if** $\sigma_2(i) \neq 0$ **then** $S_2(i,i) = S_2(i,i)/\sqrt{\sigma_2(i)}$

end

return S_1, S_2
end

Algorithm 2: Computing the symmetric scaling factors of the primal-dual matrix

3.3 $Z^T(H+D)Z$ Positive Definite, A Full-Rank

In this and the subsequent sections, we assume that the matrices have been scaled as described in section 3.2. The scaling matrices will not be mentioned explicitly.

The solution method in this case reduces to the classical reduced Hessian approach, which can be algorithmically described as follows:

1. Compute a null-space basis Z for A
   (section 3.3.1 and Algorithm 4 or 5).

Algorithm: est2Norm

Input: $H \in \Re^{n \times n}$ and the number of trials τ
Output: $h \in \Re^n_+$, an estimate of the 2-norms of the rows of H

begin

Initialization.
est $= O_{\tau \times n}$

Sampling loop.
for $t = 1$ **to** τ **do**

Generate a random vector e with entries being 1 *or* -1 *with equal probability.*
$e =$ randn$(n, 1)$
$e = (e > 0) - (e \leq 0)$

Estimate 2-norms of the rows of H for this iteration.
est$(t, :) = abs(He)'$

end

Take the root mean square of the estimates
for $i = 1$ **to** n **do** $h(i) = \frac{\|est(:,i)\|_2}{\sqrt{\tau}}$

return h
end

Algorithm 3: Demmel and Chen's algorithm to estimate the 2-norms of rows of a matrix given in operator form.

2. Solve for a particular solution of $A\delta x_\pi = r_2$
   (section 3.3.2 and Algorithm 6).

3. Solve the positive definite reduced Hessian system
   $Z^T(H+D)Z\delta x_\eta = -Z^T(r_2 + (H+D)\delta x_\pi)$
   (section 3.3.3).

4. Set $\delta x = Z\delta x_\eta + \delta x_\pi$.

5. Solve for the search direction δy using $A^T\delta y = r_2 + (H+D)\delta x$
   (section 3.3.4 and Algorithm 7).

The following sections describe implementation details for this procedure.

3.3.1 Computation of a Null-space Basis Z

Given a rectangular matrix A, its null-space basis Z can be computed by a variety of well known linear algebra techniques. All these techniques represent a trade-off between sparsity and numerical stability. For example, one may compute a QR factorization of A^T to get an orthogonal basis. The main problem with this approach is that even if the ordering strategies achieve sparsity in the factor R, the factor Q (which is a basis for the range of A^T) is likely to be dense. An implicit representation of Q via Householder matrices is of similar sparsity to R if $m = n$, but it starts becoming dense as $n - m$ increases (Theorem 1 and 3 in [GNP97]). MATLAB's `null` command uses dense SVD by default to get an orthogonal basis of the null space. The basis so obtained is again dense. Additionally, this method is computationally expensive and can be ignored for our purposes.

A computationally cheaper alternative is to use LU decomposition of A or A^T to get a sparse implicit representation of null-space basis. The basis so computed is not orthogonal, but this turns out not to be an issue in optimization when second derivatives are available. We ensure that Z is well-conditioned regardless of the condition of A. It should be noted that *Z will be implicitly represented* as a product of a few sparse matrices. This representation is memory efficient and allows for easy multiplication of a vector with either Z or Z^T.

In LU decomposition, there are two goals to be met – sparsity of the factors and their numerical stability. Both are affected by row and column ordering the matrix being factored. Ordering requirements for sparsity and stability are often conflicting, making the overall factorization difficult. In IPSOL, sparsity in the factors is promoted by a row reordering of the matrix A, while columns are permuted during the course of LU factorization to ensure

numerical stability. As explained below, the issue of stability can further be treated by observing that in the computation of a null-space basis all we need is either the L or U factor but not both. We are able to ensure that one of these factors is well-conditioned.

Two Variants of LU Factorization

Let $A \in \Re^{m \times n}$, with $m < n$, then either of the two variants of LU factorization of A can be used to compute a matrix Z such that $AZ = 0$:

- Fat LU technique: Perform LU on the "fat" matrix A:

$$PAQ = L_1 \, [U_1 \quad U_2], \tag{3.2}$$

  where P and Q are permutations, and define the null space to be

$$Z = Q \begin{bmatrix} -U_1^{-1} U_2 \\ I \end{bmatrix}. \tag{3.3}$$

- Skinny LU technique: Perform LU on the "skinny" matrix A^T:

$$PA^T Q = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} U_1, \tag{3.4}$$

  where P and Q are permutations, and define the null space to be

$$Z = P^T \begin{bmatrix} -L_1^{-T} L_2^T \\ I \end{bmatrix}. \tag{3.5}$$

P and Q both effect stability and sparsity.

From a theoretical stance, it might seem like a repetition to label the above two techniques as being different but the subtle point is that from an implementation point of view their numerical performances are not the same. A close observation of the techniques reveal that the fat technique uses the U factor to compute Z while the skinny technique relies only on the L factor. The older sparse `lu` routine in MATLAB tries to keep L well-conditioned irrespective of the condition of A. It is U that reflects the condition of A. In view of this observation, clearly the skinny-LU is desirable. It should further be noted that the UMFPACK `lu` command in MATLAB yields L and U factors that spread the condition

number of A across both the factors. The UMFPACK `lu` is unsatisfactory because to find a "good" Z, we need to reflect the condition of A either in L or in U. LUSOL [GMW87] is another sparse LU package that is suitable for null-space computations. It is likely to give a sparser Z because it finds P and Q as it goes. For maximum sparsity, threshold partial pivoting controls the condition of L (but not U); it can be used on A^T. Better stability and rank-revealing properties are provided by threshold rook pivoting, which can be used on A or A^T.

Algorithm: computeZ

Input: $A \in \Re^{m \times n}$ with $m < n$ and $\text{rank}(A) = m$
Output: $Z \in \Re^{n \times (n-m)}$ such that $AZ = 0$

begin

Compute a row permutation of A to reduce fill-in of the LU factors.
$q = \text{colamd}(A^T)$

Perform LU-factorization after permuting the rows.
The columns of A are permuted in-situ for stability in factors.
$[L, U, P] = \text{lu}(A(q, :)^T)$

Extract relevant factors.
$L_1 = L(1 : m, 1 : m)$
$L_2 = L(m + 1 : n, 1 : m)$

Implicitly store Z in terms of P, L_1 and L_2.

return Z
end

Algorithm 4: Computation of a null-space basis

An alternative way of computing the null-space matrix is discovered by observing that the presence of slacks gives A the following special structure:

$$A = \begin{bmatrix} A_{\mathscr{I}} & -I \\ A_{\mathscr{E}} & O \end{bmatrix}, \tag{3.6}$$

where $A_{\mathscr{I}}$ is the constraint matrix associated with inequality constraints and $A_{\mathscr{E}}$ with equality constraints. A null space of A is

$$Z = \begin{bmatrix} Z_{\mathscr{E}} \\ A_{\mathscr{I}} Z_{\mathscr{E}} \end{bmatrix}, \tag{3.7}$$

where $Z_{\mathscr{E}}$ is a null space basis for the equality constraint matrix $A_{\mathscr{E}}$. This approach requires LU factorization of the smaller matrix $A_{\mathscr{E}}$.

Algorithm: computeZcheaper

Input: $A_{\mathscr{E}} \in \Re^{m_{\mathscr{E}} \times n}$ with $m_{\mathscr{E}} < n$ and $\text{rank}(A_{\mathscr{E}}) = m_{\mathscr{E}}$
Input: $A_{\mathscr{I}} \in \Re^{m_{\mathscr{I}} \times n}$
Output: $Z \in \Re^{(n+m_{\mathscr{I}}) \times m_{\mathscr{E}}}$

begin

Compute a null-space basis (Algorithm 4).
$Z_{\mathscr{E}} = \text{computeZ}(\ A_{\mathscr{E}}\ )$

Implicitly store Z in terms of $A_{\mathscr{I}}$ and $Z_{\mathscr{E}}$.

return Z
end

Algorithm 5: Computation of a cheap null-space basis matrix

3.3.2 Solving for a Particular Solution of $A\delta x = r_2$

Null-space computation proceeds by factoring the constraint matrix A (Algorithm 4). These same factors can be used trivially to get a particular solution δx_π of the equation $A\delta x = r_2$ (Algorithm 6).

Algorithm: computePrt

Input: P, L, U, Q factors of $A \in \Re^{m \times n}$ $(LU = PA^TQ)$
Input: $r_2 \in \Re^m$, the residual of the constraint equations
Output: δx_π such that $A\delta x_\pi = r_2$

begin

Extract triangular portion of L.
$L_1 = L(1:m, 1:m)$

Use back substitution to solve the following system
$v^T L_1 U = r_2^T Q$
$\delta x_\pi = P^T[v; O_{(n-m)\times 1}]$

return δx_π
end

Algorithm 6: Computation of a particular solution of $A\delta x = r_2$

3.3.3 Solving the Reduced Hessian System

Under the positive definiteness assumption of the reduced Hessian, the system $Z^T(H + D)Z\delta x_\eta = -Z^T(r_2 + (H + D)\delta x_\pi)$ can be solved by direct methods like the Cholesky factorization or by iterative methods like the conjugate gradient method (CG) [HS52]. The following observations about the reduced Hessian system are strong enough to tilt the balance in favor of CG[1].

- The reduced Hessian Z^THZ is typically dense, even though individually Z and H might be sparse. Its explicit formation requires large amounts of memory and time.
- In many cases the Hessian H might be available only as an operator for forming matrix-vector products Hv.
- The desired accuracy in the solution varies across the subproblems. Initially IPSOL solves subproblems coarsely/incompletely, but later as we approach the solution, the accuracy required is much greater.

The CG method is well documented [MS06] and at this stage of exposition can be used as a black-box to compute the solution of system mentioned above to a desired level of accuracy.

3.3.4 Solving for the Dual Search Directions

Null-space computation proceeds by factoring the constraint matrix A (Algorithm 4). The same factors can be used trivially to get a particular solution of the over-determined but compatible system $A^T\delta y = r_1 + (H + D)\delta x$ (Algorithm 7).

3.3.5 Correctness of the Algorithm

By construction, the vectors δx and δy satisfy (3.1). Further, in view of the assumptions of Theorem 3.1, these equations have a unique solution. Thus δx and δy are the unique solutions of (3.1). (Of course, different approximations to δx lead to different approximations to δy.)

[1]This choice is in stark contrast with active-set based optimizers, in which direct solvers are a norm. In an active-set method, successive systems are closely related. The matrix of the current system differs from the next one by a low rank (often rank 1) update and hence the new system can be solved extremely efficiently by updating the factors of the current matrix.

Algorithm: computeDy

Input: P, L, U, Q factors of $A \in \Re^{m \times n}$ $(PA^TQ = LU)$
Input: $r \in \Re^n$
Output: $\delta y \in \Re^m$

begin

Extract the triangular portion of the L-factor
$L_1 = L(1:m, 1:m)$

Use back substitution to solve the following system,
$r = Pr$
$L_1 U v = r(1:m)$
$\delta y = Qv$

return δy
end

Algorithm 7: Solving $A^T \delta y = r$ to compute the search direction for the dual variable y

3.4 $Z^T(H + D)Z$ Indefinite, A Full-Rank

For a non-convex optimization problem, the indefiniteness of the reduced Hessian $Z^T(H + D)Z$ is not known a priori but detected during the course of solving the reduced Hessian system. Indefiniteness of the reduced Hessian is a reflection of indefiniteness of H. If indefiniteness is detected, the Hessian is suitably modified to ensure positive definiteness of the reduced Hessian. The modified primal-dual system is solved with this modified Hessian.

Conceptually, the simplest Hessian modification is addition of a positively scaled identity matrix to the original Hessian:

$$\bar{H} = H + \mu\theta I. \tag{3.8}$$

The eigenvalues of $\bar{H}$ and H are related by a positive shift:

$$\lambda(\bar{H}) = \lambda(H) + \mu\theta, \tag{3.9}$$

and so $\lambda_{\min}(\bar{H}) > \lambda_{\min}(H)$. The Hessian modification (3.8) would arise naturally if the objective function $f(x)$ were replaced by $f(x) + \frac{1}{2}\mu\theta\|x\|_2^2$.

IPSOL starts every search direction computation with the Hessian (3.8) and this Hessian is further modified when $Z^T(H+\mu\theta I+D)Z$ is judged to be indefinite. The modified Hessian

is now taken to be

$$\bar{H} = H + \mu\theta I + \sigma Z(Z^T Z)^{-2} Z^T + \gamma A^T A. \tag{3.10}$$

By appropriately choosing the parameter σ, it is possible to make the reduced Hessian $Z^T \bar{H} Z$ positive definite (Theorem 3.2), thereby ensuring the existence of a unique solution to the modified primal-dual system equation:

$$\begin{bmatrix} -(\bar{H}+D) & A^T \\ A & \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}. \tag{3.11}$$

The parameter γ is chosen to ensure that $\delta x^T \bar{H} \delta x$ is sufficiently positive (Theorem 3.3 and 3.4). We show in Theorem 3.4 that choosing γ is trivial.

3.4.1 Theoretical Framework

Existence and uniqueness of the solution of the modified primal-dual system

The following theorem guarantees that for a suitable parameter σ, $Z^T(\bar{H}+D)Z \succ 0$. Since A is already assumed to have full rank, (3.11) has a unique solution with that value of σ.

Theorem 3.2. *Let Z be a null-space basis for A. If the Hessian H is modified as in (3.10), then $\forall \theta \geq 0$ there exists a $\sigma \geq 0$ such that $Z^T(\bar{H}+D)Z \succ 0$.*

Proof. Notice that $Z^T(\bar{H}+D)Z = Z^T(H+D+\mu\theta I)Z + \sigma I$. Thus by setting

$$\sigma = \begin{cases} -\lambda_{\min}(H+\mu\theta I+D)+\epsilon & \text{if indefiniteness is detected} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

for some $\epsilon > 0$ it is ensured that $Z^T(\bar{H}+D)Z \succ 0$. □

An advantage of choosing σ as in (3.12) is that for positive-definite systems the contribution of the term $Z(Z^TZ)^{-2}Z$ in $\bar{H}$ is zero. However, in view of the contingencies arising out of Theorem 3.4, σ is sometimes chosen as

$$\sigma = \begin{cases} -\lambda_{\min}(H+\mu\theta I+D)+\epsilon & \text{if indefiniteness is detected} \\ \epsilon & \text{otherwise} \end{cases} \tag{3.13}$$

for some $\epsilon > 0$. With this choice of σ, the term $Z(Z^TZ)^{-2}Z^T$ in $\bar{H}$ (3.10) is always non-zero, thereby making the search direction computations costlier. However, it is only in rare

cases that a search direction defined with σ as in (3.12) would fail to satisfy the inequality

$$\delta x^T \bar{H} \delta x \geq \epsilon \|\delta x\|_2^2. \tag{3.14}$$

Consequently, the computationally cheaper choice (3.12) is tried first in the implementation. Only if the resulting search direction δx fails to satisfy (3.14) is δx recomputed with σ as in (3.13).

Ensuring sufficient descent and sufficient negative curvature

The following theorems provide a guideline for choosing a scalar γ in (3.10) so that the solution δx of (3.11) is such that $\delta x^T(\bar{H} + D)\delta x$ is positive and bounded away from zero.

Theorem 3.3. *The δx component of the solution of the modified primal-dual system (3.11) is independent of the parameter γ.*

Proof. The proof is by construction. Since A has full row rank, the equation $A\delta x = r_2$ is consistent. Let δx_π be a particular solution of this system, which can clearly be chosen independent of γ. Also, notice that the reduced Hessian is independent of γ:

$$\begin{aligned} Z^T(\bar{H} + D)Z &= Z^T(H + \mu\theta I + \sigma Z(Z^T Z)^{-2} Z^T + \gamma A^T A + D)Z \\ &= Z^T(H + \mu\theta I + \sigma Z(Z^T Z)^{-2} Z^T + D)Z \\ &= Z^T(H + \mu\theta I + D)Z + \sigma I. \end{aligned}$$

Thus, the null-space component δx_η is also independent of γ. Hence, $\delta x = \delta x_\pi + \delta x_\eta$ is independent of γ. □

Theorem 3.4. *If the δx component of the solution of the modified primal-dual system (3.11) is computed with σ chosen as in (3.13) then there exists a $\gamma \geq 0$ such that δx satisfies the inequality $\delta x^T(\bar{H} + D)\delta x \geq \epsilon \|\delta x\|_2^2$.*

Proof. In view of the last theorem, δx is independent of γ. Consider two cases.

1. Case $r_2 = 0$: In this case, $\delta x_\pi = 0$ as A has full rank and δx_π satisfies $A\delta x_\pi = 0$. The solution δx just consists of a component in the null space of the constraint matrix:

$\delta x = \delta x_\eta = Zp$ for some vector p. Then

$$\begin{aligned}
\delta x^T(\bar{H}+D)\delta x &= \delta x_\eta^T(\bar{H}+D)\delta x_\eta \\
&= p^T Z^T(H+\mu\theta I+\sigma Z(Z^TZ)^{-2}Z^T+\gamma A^TA+D)Zp \\
&= p^T Z^T(H+\mu\theta I+D+\sigma I)Zp \\
&= \delta x^T(H+\mu\theta I+D+\sigma I)\delta x \\
&\geq (\lambda_{\min}(H+\mu\theta I+D)+\sigma)\|\delta x\|_2^2.
\end{aligned}$$

Defining σ as in (3.13) ensures that $\lambda_{\min}(H+\mu\theta I+D)+\sigma \geq \epsilon$ and the result follows immediately. Hence, $\gamma = 0$ is suitable for this case.

2. Case $r_2 \neq 0$: Since $A\delta x = r_2$, in this case $\delta x^T A^T A\delta x = \|r_2\|_2^2$.

$$\begin{aligned}
\delta x^T(\bar{H}+D)\delta x &= \delta x^T(H+\mu\theta I+\sigma Z(Z^TZ)^{-2}Z^T+\gamma A^TA+D)\delta x \\
&= \delta x^T(H+D+\mu\theta I+\sigma Z(Z^TZ)^{-2}Z^T)\delta x+\gamma\|r_2\|_2^2
\end{aligned}$$

Clearly, setting

$$\gamma = \min\left(0, \frac{\epsilon\|\delta x\|_2^2 - \delta x^T(H+D+\mu\theta I+\sigma Z(Z^TZ)^{-2}Z^T)\delta x}{\|r_2\|_2^2}\right)$$

is a satisfactory choice.

□

The next theorem helps in finding a good direction of negative curvature.

Theorem 3.5. *If the CG method detects indefiniteness in the reduced Hessian system*

$$Z^T(H+\mu\theta I+D)Zp = Z^T(r_1+(H+\mu\theta I+D))\delta x_\pi$$

then its current approximation to p is a direction of sufficient negative curvature for the original reduced Hessian $Z^T(H+D)Z$.

Proof. Let α_k^{cg} denote the step at the kth CG iteration and p_k be the associated approximation to p. If CG detects indefiniteness at the kth iteration, it terminates with p_k that satisfies the inequality $p_k^T(Z^T(H+\mu\theta I+D)Z)p_k < 0$. It is easy to see that $p_k^T Z^T(H+D)Zp_k < -\mu\theta\|p_k\|_2^2$. Thus p_k is a direction of sufficient negative curvature of the unmodified system. □

The last theorem also suggests how to detect indefiniteness in a symmetric linear system of equations. In view of the conclusions of Theorem 3.5, the CG method can be modified as described in Algorithm 8.

Efficient computation of $Z(Z^TZ)^{-2}Z^T$ with δx and δx_π

Using the Hessian modification described in (3.10) to solve the modified primal-dual system (3.11) by a null-space approach involves multiplication of the vectors δx and δx_π by $Z(Z^TZ)^{-2}Z^T$, where δx_π is a particular solution of $A\delta x = r_2$. The following theorems show that these multiplications can be done efficiently without computing $Z(Z^TZ)^{-2}Z^T$.

Theorem 3.6. *Let $A \in \Re^{m\times n}$ $(m < n)$ be a full-rank matrix. A vector $\delta x \in \Re^n$ is a minimum-norm solution of $Ax = b$ iff it is orthogonal to the null space of A.*

Proof. The if part: the minimum-norm solution of $Ax = b$ is given in closed form as $\delta x_\perp = A^T(AA^T)^{-1}b$. Let Z be a null-space basis for A. The result follows from observing that $Z^T\delta x_\perp = Z^TA^T(AA^T)^{-1}b = 0$. □

An obvious way of constructing the minimum-norm solution is to run LSQR [PS82] on the system of equations $A\delta x = r_2$, but if A were ill-conditioned then the convergence would be slow. The following result makes this process more efficient if Z is available and is calculated using LU factors of A.

Theorem 3.7. *Let Z be a null-space basis for a full-rank matrix $A \in \Re^{m\times n}$ $(m < n)$. If Z is computed using LU factors of A (Algorithm 4) then the minimum-norm solution of $A\delta x = r_2$ can be efficiently computed by two back-solves and by solving a least squares problem involving Z.*

Proof. Since the LU factors of A are readily available, they can be used to construct a particular solution of $A\delta x_\pi = r_2$ by back substitution (see Algorithm 6). This solution can be decomposed into a component lying in the null space of A (say $\delta x_\parallel$) and another component lying perpendicular to this null space $\delta x_\perp$):

$$\delta x_\pi \;=\; \delta x_\parallel + \delta x_\perp \;=\; \delta x_\parallel + Zp.$$

Hence,

$$Z^T\delta x_\pi \;=\; Z^T(\delta x_\perp + Zp) \;=\; Z^TZp.$$

Algorithm: Modified Conjugate Gradient

Input: $A \in S\mathbb{R}^{n\times n}, b \in \mathbb{R}^n$
Output: x, dn, lcgFail

begin

$r_1 = b$
$\beta_1 = 0$
$x_1 = 0$
lcgFail = false
$dn = []$

for *k = 1, 2, ...* **do**

Computation of the search directions.
$\gamma_k = r_k^T r_k$
$\beta_k = \gamma_k / \gamma_{k-1}$
$p_k = r_k + \beta_k p_{k-1}$
$v_k = Ap_k$

Determination of the step length.
$\sigma_k = p_k^T v_k$
$\alpha_k^{\text{cg}} = \gamma_k / \sigma_k$

Detection of indefiniteness in A.
if $\alpha_k^{cg} \leq 0$ **then**
$dn = p_k$
lcgFail = true
break
end

Update of the iterates.
$x_{k+1} = x_k + \alpha_k^{\text{cg}} p_k$
$r_{k+1} = r_k - \alpha_k^{\text{cg}} v_k$

end

$x = x_k$
end

return x, dn, lcgFail

Algorithm 8: Modified Conjugate Gradient

So, the vector p is a solution of $Z^T Zp = Z^T \delta x_\pi$ or the least squares problem,

$$\min_p \|Zp - \delta x_\pi\|^2,$$

which can be solved by LSQR. Then $\delta x_\perp = \delta x_\pi - Zp$ by the above theorem is the minimum-norm solution of the equation $A\delta x = r_2$. □

Since Z is likely to be well-conditioned even if A is not, the above solution procedure is more efficient than directly running LSQR on $A\delta x = r_2$. In view of the above results, by cheaply modifying a given particular solution δx_π to a minimum-norm solution, we ensure that the expression $Z(Z^T Z)^{-2} Z^T \delta x_\pi$ reduces to zero. This is the idea behind Algorithm 9.

Algorithm: orthogonalizePrt

Input: $A \in \Re^{m\times n}$ with $m < n$ and rank$(A) = m$
Input: $Z \in \Re^{n\times(n-m)}$ null-space matrix of A ($AZ = 0$).
Input: $r_2 \in \Re^m$ the constraint residual.
Input: $\delta x_\pi \in \Re^n$, any particular solution of $A\delta x = r_2$

Output: $\delta x_\perp$ the minimum-norm solution of $A\delta x = r_2$

begin

Use LSQR to get a least square solution of the equation $Zp = \delta x_\pi$

Purge δx_π *of the component lying in the null-space of A:* $\delta x_\perp = \delta x_\pi - Zp$

return $\delta x_\perp$

end

Algorithm 9: Constructing the minimum-norm solution of $A\delta x = r_2$

If the particular solution is so chosen, then the following theorem shows how to compute $Z(Z^T Z)^{-2} Z^T \delta x$, where δ_x is a solution of (3.11).

Theorem 3.8. *Let Z be a null-space basis for a full rank matrix $A \in \Re^{m\times n}$ ($m < n$). The vector $v = Z(Z^T Z)^{-2} Z^T \delta x$ can be computed efficiently by solving a minimum-norm problem involving Z^T.*

Proof. The solution δx can be decomposed as $\delta x = \delta x_\perp + Zp$. Then

$$\begin{aligned} v &= Z(Z^TZ)^{-2}Z^T\delta x \\ &= Z(Z^TZ)^{-2}Z^T(\delta x_\perp + Zp) \\ &= Z(Z^TZ)^{-1}p, \end{aligned}$$

which is the minimum-norm solution of $Z^Tv = p$. □

3.4.2 Solution Strategy

With the above theoretical tools in place, the algorithm can be stated as follows.

1. Compute a null-space basis Z for A
   (section 3.3.1 and Algorithm 4 or 5).

2. Solve for a particular solution of $A\delta x_\pi = r_2$
   (section 3.3.2 and Algorithm 6).

3. Attempt to solve the possibly indefinite system
   $Z^T(H + \mu\theta I + D)Zp = -Z^T(r_2 + (H + \mu\theta I + D)\delta x_\pi)$
   (section 3.3.3 and Algorithm 8).

4. If indefiniteness is not detected in the above system, set $\delta x = Zp + \delta x_\pi$. Otherwise,

   - Compute σ, an estimate of the minimum eigenvalue of $Z^T(H + \mu\theta I + D)Z$.
   - Compute $\delta x_\perp$
     (Theorem 3.7 and Algorithm 9).
   - Attempt to solve the possibly indefinite system
     $(Z^T(H + \mu\theta I + D)Z + \sigma I)p = -Z^T(r_2 + (H + \mu\theta I + D)\delta x_\perp)$
     (section 3.3.3 and Algorithm 8).
   - Set $\delta x = Zp + \delta x_\perp$.

5. Compute γ to ensure $\delta x^T(\bar{H} + D)\delta x$ is sufficiently positive
   (Theorem 3.4).

6. Compute $v = Z(Z^TZ)^{-2}Z^T\delta x$ by solving for the minimum-norm solution of $Z^Tv = p$
   (Theorem 3.8).

7. Solve for the dual search direction by solving the compatible system

$$A^T \delta y = r_2 + (H + \mu\theta I + D + \gamma A^T A)\delta x + \sigma v$$

(section 3.3.4 and Algorithm 7).

The direction of negative curvature computed by Algorithm 8 may not be a good one. However, it may be improved by application of a few steps of the Lanczos algorithm. For positive definite systems, it can be shown that the conjugate gradient algorithm is a memory efficient formulation of the Lanczos algorithm [MS06]. In this case, the sequence of iterates generated from either algorithm be obtained from the other. In the implementation, one may store the necessary information to switch seamlessly from the conjugate gradient algorithm to the Lanczos algorithm, in case indefinitenes is detected. This avoids losing the information about the subspace explored so far. However, this approach defeats the purpose of having a memory efficient formulation of Lanczos algorithm. As a compromise, we store the Lanczos vectors and the tridiagonal system for the first few conjugate gradient iterations.

3.5 $Z^T(H + D)Z$ Indefinite, A Rank Deficient

In the simplest case, rank deficiency in A can arise because of an accidental duplication of one or more constraints. A rank-deficient constraint matrix A can give rise to the following difficulties:

- The presence of rank deficiency in the active constraint matrix in a neighborhood of a local minimizer violates the assumptions under which the first-order optimality conditions hold. The Lagrange multipliers at such a minimizer are unbounded. A near rank deficiency in a neighborhood of a solution gives rise to large dual variable steps that may result in a poor rate of convergence.

- Rank deficiency in A at an intermediate iterate can additionally give rise to an incompatibility in the primal-dual system (3.11). In this case there is no solution to (3.11), causing a complete breakdown of the search direction algorithm.

Rank deficiency in A can be detected during the computation of its null-space basis (Algorithm 4). The LU factorization of a (near) rank deficient A has (small) zero elements on the diagonal of the U factor. The indices of columns with $U_{jj} \approx 0$ correspond to

the permuted indices of the dependent constraints in A. By the addition of elastic slack-variables in these constraints the constraint matrix is made full rank and the solution strategy proceeds without modification as described in last section. The modified primal-dual system of equations with elastic variables is derived in Appendix A.

In an interior point method, even with elastic variables, rank deficiency can show up near a non-degenerate solution. The scaling procedure described in section 3.2 can introduce ill-conditioning in A while improving the overall condition of the primal-dual matrix. This case can be remedied in part by observing that near a minimizer with rank-deficient active constraints, $\delta x \to 0$ and $\delta y \to \infty$. This motivates computing $\delta x_\perp$ and δy as solutions of the regularized least squares problems

$$\min_{\delta x_\pi} \|r_2 - A\delta x_\pi\|_2^2 + \|\delta x_\pi\|_2^2, \text{ and} \tag{3.15}$$

$$\min_{\delta y} \|r_1 + (\bar{H} + D)\delta x - A^T \delta y\|_2^2 + \mu\|\delta y\|_2^2, \tag{3.16}$$

while δx_η is obtained from the usual reduced Hessian system. Both these regularized least squares problems can be stably solved by LSQR. With these modifications, the algorithm described in the last section does not break down in the presence of rank deficient constraints.

3.6 Conclusion

Once δx and δy (and possibly dn) have been computed, they can be used trivially to compute search directions for the remaining variables:

$$\begin{aligned}
\delta t_{l_k} &= E_l \delta x_{l_k} - r_l \\
\delta t_{u_k} &= -E_u \delta x_{u_k} - r_u \\
dnt_{l_k} &= E_l dn \\
dnt_{u_k} &= -E_u dn \\
\delta z_{l_k} &= (r_{pcl} - z_{l_k} .* \delta t_{l_k})/t_{l_k} \\
\delta z_{u_k} &= (r_{pcu} - z_{u_k} .* \delta t_{u_k})/t_{u_k}.
\end{aligned} \tag{3.17}$$

In the next chapter, we describe how to perform a linesearch along this set of search directions to decide how far to move along these directions.

Chapter 4

Merit Function Based Linesearch

4.1 Introduction

The search directions given by the algorithm in chapter 3 dictate the direction to move along. An arbitrary step taken in that direction may result in a diverging sequence. Linesearch algorithms define a step along the search directions that ensures global convergence of the algorithm from an arbitrary starting point. The other globalization approach in use today is the trust-region approach [CGT00]. In trust-region algorithms, the search direction and the step-length computations are combined into one. In either of these approaches, the goodness of the steps is determined by the use of a merit function or by a filter method [FGL+02]. This chapter explores the issues of merit functions together with the step-length algorithm.

4.2 Merit Functions and Linesearch Algorithms

SQP algorithms are infeasible point algorithms. The intermediate iterates generated by them need not satisfy nonlinear constraints. The role of a merit function is to ensure a balance between infeasibility reduction and optimality, thereby ensuring global convergence. Merit functions are a weighted sum of the constraint infeasibility and the objective function values. The weight on infeasibility is called a penalty parameter, denoted by ρ. As stated above, they are used to evaluate the "goodness" of steps along the search direction. A linesearch algorithm is used to choose a step that gives a "sufficient decrease" in the value of the merit function. The topic of merit functions has been well-researched in the field of nonlinear optimization. The choice of a particular merit function together with the rules

for updating the merit-penalty parameter has a significant impact on the performance of an SQP implementation. A good merit function should have the following characteristics:

- Exactness: A merit function is said to be exact if for finite (but possibly large) values of the penalty parameter every (local) minimizer of the original optimization problem is also a (local) minimizer of the merit function[1].
- Differentiability: Differentiability enables us to construct a surrogate of the merit function using simple continuous functions. This surrogate may be used to get a good approximation of a local minimizer of the merit function during the linesearch procedure. Absence of differentiability forces the linesearch algorithm to rely on naive and inefficient backtracking procedures.
- Fast rate of convergence: A necessary and sufficient condition for superlinear convergence of Newton's method is that unit steps (or steps converging to unit steps) be taken in the vicinity of the solution. There are merit function that can reject unit steps in close vicinity of a solution (e.g., the l_1 merit function), thereby hindering the superlinear rate of convergence. This effect was first brought to light by Maratos [MM79] and hence named after him. Unit steps in the neighborhood of the solution are more challenging for log-barrier methods than they are for active-set methods. In barrier methods, a change is made to a variable even if it is very close to a bound. In active-set methods, the same variable would be on its bound and the search direction in that variable would be zero.

A few representative merit functions for equality-constrained problems are listed in Table 4.1. The last example is the augmented Lagrangian merit function.

4.3 Use of Augmented Lagrangian Merit Function in IPSOL

In IPSOL, the augmented Lagrangian merit function is used. This merit function does a search in both the primal and dual variables, is differentiable, and does not suffer from the Maratos effect. Further, its successful use in SNOPT [GMS02, GMS05] establishes its worth in practice. It is for these reasons that this merit function was chosen for this project. The augmented Lagrangian merit function is used to ensure global convergence of

[1]It should be noted that the converse of this statement is *not* true. Irrespective of the value of the penalty function, a merit function may have one or more (local) minimizers that are infeasible for the original optimization problem.

Merit function	Nature	Analytic properties	Maratos Effect
$f(x) + \rho\|c_{\mathscr{E}}(x)\|_1$	Exact [Pie69]	Non-differentiable at points that are feasible w.r.t. *any* set of constraints.	Yes [MM79]
$f(x) + \rho\|c_{\mathscr{E}}(x)\|_2$	Exact	Non-differentiable at points that are feasible w.r.t. *all* constraints.	-
$f(x) + \frac{\rho}{2}\|c_{\mathscr{E}}(x)\|_2^2$	Inexact	C1-differentiable at *all* points.	-
$f(x) - y^T c_{\mathscr{E}}(x) + \frac{\rho}{2}\|c_{\mathscr{E}}(x)\|_2^2$	Inexact	C1-differentiable at *all* points.	No

Table 4.1: Representative merit functions for equality constrained problems

the equality-constrained problem (2.2) for a *fixed* value of μ. The augmented Lagrangian of (2.2) is defined as [2]

$$\begin{aligned} M_{\mathscr{L}}(\overline{x}, t_l, t_u, y, z_l, z_u) := \quad & f(\overline{x}) - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u)] \\ & -y^T c(\overline{x}) - z_l^T (E_l \overline{x} - t_l - l) - z_u^T (-E_u \overline{x} - t_u + u) \\ & +\rho(\|c(\overline{x})\|_2^2 + \|E_l \overline{x} - t_l - l\|_2^2 + \| - E_u \overline{x} - t_u + u\|_2^2). \end{aligned} \tag{4.1}$$

The search directions derived in the last chapter might not be directions of descent for the augmented Lagrangian merit function. However, it is always possible to make them a direction of descent for a sufficiently large but finite value of ρ [MP95]. For details of optimally selecting and updating ρ, please see [Eld92].

4.4 Linesearch Termination Criteria

The merit function is not minimized along a specific search direction. Instead, an approximation to a minimizer is obtained by satisfying certain termination criteria. How tight the criteria are made depends on the effort to evaluate the problem functions. Given a merit functions (and/or their gradients) and a descent direction, inexact linesearch algorithms are designed to give a step-length along the descent direction at which there occurs a "sufficient decrease" in the merit function. Additionally, these algorithms ensure that the steps taken are not arbitrarily close to zero so that reasonable progress is made toward the solution. Such conditions guarantee convergence to a stationary point of the merit function for a wide

[2] We wish to point out that SNOPT does not include linear or bound constraints as a part of the definition of the augmented Lagrangian merit function. In SNOPT, the initial point is chosen to satisfy these constraints and subsequent steps ensure that they remain satisfied. In IPSOL, the initial point may not satisfy these constraints (please see section 2.5)

variety of search directions.

These requirements are articulated in terms of a variety of line-search termination criteria, an elaborate discussion of which maybe found in [OR70]. If

$$\phi(\alpha) \equiv M_{\mathscr{L}}(\overline{x} + \alpha\delta\overline{x}, t_l + \alpha\delta t_l, t_u + \alpha\delta t_u, y + \alpha\delta p_y, z_l + \alpha\delta z_l, z_u + \alpha\delta z_u) \tag{4.2}$$

then some of the well known termination criteria can be written as follows.

- **Armijo condition [Arm66]:** Let α_0 be the maximum permissible trial step. Termination occurs at a trial step $\alpha = \beta\alpha_0$ if

$$\boxed{\phi(\alpha) - \phi(0) \leq \beta\alpha_0\phi'(0)} \tag{4.3}$$

  for some chosesn $\beta \in (0, 1)$. This condition prevents the steps from being too large. Since $\phi'(0) < 0$, a sufficiently small value of β clearly satisfies the equation for any α_0.

- **Strong Wolfe conditions:** Given two constants σ and η s.t. $0 < \sigma \leq \eta < 1$, this termination criterion is satisfied by α if the following conditions hold:

$$\boxed{\begin{array}{rcl} \phi(\alpha) & \leq & \phi(0) + \sigma\alpha\phi'(0) \\ |\phi'(\alpha)| & \leq & \eta|\phi'(0)|. \end{array}} \tag{4.4}$$

  The first condition ensures a limit on the size of the step, while the second condition is usually the one that governs how the step is chosen. If η is small, the step can be made arbitrarily close to a minimizer. In the literature, the first condition is called the sufficient-decrease condition while the second one is referred to as the curvature condition. Moré and Thuente have described an algorithm for finding a step satisfying these conditions [MT94].

- **1D-Gamma conditions [GMSW79]:** A simple modification to the Strong Wolfe conditions simplifies the step-length algorithm. The modification is to accept the maximum permissible step if it satisfies the sufficient decrease condition:

$$\boxed{\begin{array}{rcl} \phi(\alpha_{\max}) & \leq & \phi(0) + \sigma\alpha_{\max}\phi'(0) \\ & OR & \\ \phi(\alpha) & \leq & \phi(0) + \sigma\alpha\phi'(0) \\ |\phi'(\alpha)| & \leq & \eta|\phi'(0)| \end{array}} \tag{4.5}$$

The set of values of α satisfying these conditions is denoted by $\Gamma 1D(\sigma, \eta)$. These conditions are not satisfactory at a saddle-point where $\phi'(0) = 0$ and there is a direction of negative curvature.

- **2D-Gamma conditions [FM93]:** 1D-Gamma conditions are easily extended to the case when second derivative information is used and we have a direction of negative curvature. In this case

$$\begin{aligned}\phi(\alpha) \quad \equiv \quad & M_{\mathscr{L}}(\overline{x} + \alpha^2\delta\overline{x} + \alpha dn, y + \alpha^2\delta p_y, t_l + \alpha^2\delta t_l + \alpha dnt_l, t_u + \alpha^2\delta t_u + \alpha dnt_l, \\ & z_l + \alpha^2\delta z_l, z_u + \alpha^2\delta z_u).\end{aligned} \tag{4.6}$$

  - if $\phi''(0) \geq 0$, use the 1D gamma conditions (4.5).
  - else if $\phi''(0) < 0$, use the following conditions, which are essentially the 1D-gamma conditions modified to incorporate second-derivative information of ϕ:

$$\boxed{\begin{array}{lcl} \phi(\alpha_{\max}) & \leq & \phi(0) + \sigma[\alpha_{\max}\phi'(0) + \frac{1}{2}\alpha_{\max}^2\phi''(0)] \\ & OR & \\ \phi(\alpha) & \leq & \phi(0) + \sigma[\alpha\phi'(0) + \frac{1}{2}\alpha^2\phi''(0)] \\ |\phi'(\alpha)| & \leq & \eta|\phi'(0) + \alpha\phi''(0)| \end{array}} \tag{4.7}$$

  The set of values of α satisfying these conditions is denoted by $\Gamma 2D(\sigma, \eta)$.

There exist minor variations on each of these conditions, like the Armijo-Goldstein conditions and the Weak Wolfe conditions [Wol69, Wol71].

4.5 Intuition Behind the Linesearch Algorithm

In IPSOL, we use the 1D and 2D Gamma conditions. As mentioned, their key advantage over the Wolfe conditions is that our linesearch algorithm is conceptually much simpler than that of Moré and Thuente [MT94].

The underlying intuition is based on the observation that the curvature condition is satisfied by any local minimizer of $\phi(\alpha)$. Using this observation, it is trivial to show that an interval $[\alpha_l, \alpha_u]$ at whose endpoints $\phi(\alpha)$ satisfies conditions 1A or 1B in Table 4.2 contains a point $\alpha \in \Gamma 1D(\sigma, \eta)$. Similarly, an interval $[\alpha_l, \alpha_u]$ at whose endpoints $\phi(\alpha)$ satisfies conditions 2A or 2B contains a point $\alpha \in \Gamma 2D(\sigma, \eta)$.

The proposed linesearch algorithm is based on finding a sequence of such nested intervals.

Type A interval of uncertainty		Type B interval of uncertainty	
1A.	$\phi(0) + \sigma\alpha_u\phi'(0) \leq \phi(\alpha_u)$ $\phi(\alpha_l) \leq \phi(0) + \sigma\alpha_l\phi'(0)$ $\phi'(\alpha_l) \leq \eta\phi'(0) \leq 0$	1B.	$\phi'(\alpha_u) \geq 0$ $\phi(\alpha_l) \leq \phi(0) + \sigma\alpha_l\phi'(0)$ $\phi'(\alpha_l) \leq \eta\phi'(0) \leq 0$
2A.	$\phi(0) + \sigma[\alpha_u\phi'(0) + \frac{1}{2}\alpha_u^2\phi''(0)] \leq \phi(\alpha_u)$ $\phi(\alpha_l) \leq \phi(0) + \sigma[\alpha_l\phi'(0) + \frac{1}{2}\alpha_l^2\phi''(0)]$ $\phi'(\alpha_l) \leq \eta[\phi'(0) + \alpha_l\phi''(0)] \leq 0$	2B.	$\phi'(\alpha_u) \geq 0$ $\phi(\alpha_l) \leq \phi(0) + \sigma[\alpha_l\phi'(0) + \frac{1}{2}\alpha_l^2\phi''(0)]$ $\phi'(\alpha_l) \leq \eta[\phi'(0) + \alpha_l\phi''(0)] \leq 0$

Table 4.2: Types 1A or 1B guarantee the existence of an interior point α satisfying (4.5) while types 2A or 2B guarantee the existence of an interior point α of (4.7).

4.6 Linesearch Algorithm

A broad overview of the linesearch algorithm is stated below.

- **Initialize the interval of uncertainty:** If an initial step $\alpha = 1$ satisfies the sufficient decrease condition, it is accepted, else the initial interval of uncertainty is set to [0, 1]. In this case, [0, 1] is an interval of type A.
- **Trial step generation:** Using safeguarded polynomial or special interpolation, a trial step $\alpha_{trial} \in [\alpha_l, \alpha_u]$ is generated.
- **Update the interval of uncertainty:** Given an interval $[\alpha_l, \alpha_u]$, replace either α_l or α_u with the trial step α_{trial}, so that the new interval is either of type A or B.

The last two steps are repeated until a suitable step is found. This entire process is expressed in Algorithm 10. An elaborate discussion of each of these steps is given in the following subsections.

4.6.1 Initialize the Interval of Uncertainty

The Dennis-Moré theorem [NW06] states that a sufficient condition for asymptotic superlinear convergence of a nonlinear optimization algorithm is that the asymptotic steps approximate the pure Newton steps. Thus, the initial trial step is always chosen to be of

Input: $\phi(*), \alpha_{max}$
Output: α^*

begin
  $\alpha_{trial} \leftarrow \min(1, \alpha_{max})$
  if α_{trial} *gives sufficient descent* **then**
    $\alpha^* \leftarrow \alpha_{trial}$
  else
    $[\alpha_l, \alpha_u] \leftarrow [0, \alpha_{trial}]$
    $\alpha_{trial} \leftarrow$ Safeguarded Interpolation
    while $\alpha_{trial} \in \Gamma(\sigma, \eta)$ **do**
      $[\alpha_l, \alpha_u] \leftarrow$ Update the Interval of Uncertainty
      $\alpha_{trial} \leftarrow$ Safeguarded Interpolation
    end
    $\alpha^* \leftarrow \alpha_{trial}$
  end
  return α^*
end

Algorithm 10: Linesearch

unit length. If it satisfies the sufficient decrease condition, we accept it. By definition, this step is not small, so there is no need to enforce the curvature condition. However, this step may fail to satisfy the sufficient-decrease conditions. In this case we initialize the interval of uncertainty as $[0, 1]$. This interval is guaranteed to have an $\alpha \in \Gamma 1\mathrm{D}(\sigma, \eta)$ or $\alpha \in \Gamma 2\mathrm{D}(\sigma, \eta)$.

4.6.2 Trial Step Generation

Given an interval $[\alpha_l, \alpha_u]$ guaranteed to contain an $\alpha \in \Gamma(\sigma, \eta)$, the process generates an $\alpha^* \approx \alpha$. The approximation can be generated in any of the following ways.

- **Bisection:** The approximation to α is taken as the mid-point of the interval of uncertainty $[\alpha_l, \alpha_u]$. Repeated application of bisection is guaranteed to reduce the length of the interval of uncertainty to a given level in a finite number of iterations, but the total number of iterations might end up being large.

- **Polynomial interpolation:** If the merit function is well behaved, then a polynomial is often a good local approximation to it. In this approach one uses the function and derivative information available at the end points of the interval of uncertainty to construct a local polynomial approximation of the merit function. This approximation

is usually a quadratic or cubic polynomial, whose minimizer is easy to find. This minimizer lies in the interval $[\alpha_l, \alpha_u]$ and is taken as an approximation to α.

- **Special interpolation:** This is a generalization of the polynomial interpolation method to cases where the merit function is affected by logarithmic singularities in the merit function. This method is polynomial interpolation but with the addition of a logarithmic singularity to the polynomial interpolant. This singularity is added for approximating the logarithmic singularity in the merit function. An extensive discussion of these methods maybe found in [MW94].

Safeguarded interpolation combines a guaranteed method (like bisection) with a fast converging method (like interpolation) to get the best of both worlds.

4.6.3 Update the Interval of Uncertainty

It should be noted that the linesearch iteration starts out with an initial interval of type A. Depending on the slope and value of function ϕ at α_{trial}, we replace one end-point of the current interval with either α_l or α_u in such a way that the resulting interval has at least one $\alpha \in \Gamma(\sigma, \eta)$. Algorithm 11 takes an interval of type A or B and returns a nested interval of either type A or type B.

Input: $\phi(*), \alpha_l, \alpha_u, \alpha_{trial}$
Requires: $[\alpha_l, \alpha_u]$ *is an interval of type A or B.*
Requires: α_{trial} *is such that either the sufficient descent or the curvature condition is violated at it.*
Output: α_l, α_u

begin
  if α_{trial} *does not give sufficient descent* **then**
    $\alpha_u \leftarrow \alpha_{trial}$
  else if $\phi'(\alpha_{trial}) \geq 0$ **then**
    $\alpha_u \leftarrow \alpha_{trial}$
  else
    $\alpha_l \leftarrow \alpha_{trial}$
  end
  return α_l, α_u
end

Algorithm 11: Update the Interval of Uncertainty

4.7 Updating the Variables

The three steps of the linesearch algorithm are performed until a suitable step-length satisfying the appropriate descent and curvature conditions is found. This step-length is used to update the variables. If second derivatives are not being used then variables are updated as

$$\begin{array}{rcl} x_{k+1} & = & x_k + \alpha^* \delta x_k \\ y_{k+1} & = & y_k + \alpha^* \delta y_k \\ t_{l_{k+1}} & = & t_{l_k} + \alpha^* \delta t_{l_k} \\ t_{u_{k+1}} & = & t_{u_k} + \alpha^* \delta t_{u_k} \\ z_{l_{k+1}} & = & z_{l_k} + \alpha^* \delta z_{l_k} \\ z_{u_{k+1}} & = & z_{u_k} + \alpha^* \delta z_{u_k}, \end{array} \tag{4.8}$$

otherwise they are updated as

$$\begin{array}{rcl} x_{k+1} & = & x_k + \alpha^{*2} \delta x_k + \alpha^* dn \\ y_{k+1} & = & y_k + \alpha^{*2} \delta y_k \\ t_{l_{k+1}} & = & t_{l_k} + \alpha^{*2} \delta t_{l_k} + \alpha^* dnt_{l_k} \\ t_{u_{k+1}} & = & t_{u_k} + \alpha^{*2} \delta t_{u_k} + \alpha^* dnt_{u_k} \\ z_{l_{k+1}} & = & z_{l_k} + \alpha^{*2} \delta z_{l_k} \\ z_{u_{k+1}} & = & z_{u_k} + \alpha^{*2} \delta z_{u_k}. \end{array} \tag{4.9}$$

4.8 Conclusion

This chapter concludes the algorithmic description of IPSOL. In the next chapter we present the performance of the software on a set of test problemss.

Chapter 5

Numerical Results

5.1 Introduction

IPSOL has been implemented in MATLAB. This chapter summarizes the performance of IPSOL on a subset of the CUTEr test problem set [GOT03]. IPSOL's performance is also compared to that of the commercial optimizer LOQO [Van99] and open source optimizer IPOPT [WB06].

The choice of the initial point was described in section 2.5. The strategy for decreasing μ from one subproblem to another is described in section 5.2. Section 5.3 describes the various termination criteria used in IPSOL. The interface of IPSOL to the CUTEr test environment is outlined in section 5.5. The performance of IPSOL on small and large-scale problems is presented in sections 5.5 and 5.6 respectively. Finally, in section 5.7 IPSOL's performance is compared with the two solvers mentioned above.

5.2 Decreasing μ from One Subproblem to Another

IPSOL monotonically decreases μ from one subproblem to the other (each major iteration). It can be shown that near the solution,

$$\|x^*(\mu) - x^*\|_2 \sim O(\mu) \text{ [FM90]}. \tag{5.1}$$

Thus, for ensuring the quadratic convergence of the sequence $\{x^*(\mu_k)\}$, the sequence $\{\mu_k\}$ should also quadratically converge to zero. Away from the solution, μ may be decreased

linearly. The overall update rule is

$$\mu_{k+1} = \begin{cases} \mu_k^2 & \text{if } \mu_k < \theta \\ \beta\mu_k & \text{otherwise.} \end{cases}$$

The default value for θ is 10^{-2} while for β it is $\frac{1}{10}$.

5.3 Termination Criteria

All the termination criteria in IPSOL are expressed in terms of the inf-norm of the normalized residuals of equations (2.4). The normalization is done by dividing the actual residuals by the primal and the dual variables:

$$\begin{aligned} \bar{r}_L &= \tfrac{r_L}{(1+\|x\|_\infty)} \\ \bar{r}_{pcl} &= \tfrac{r_{pcl}}{(1+\|y\|_\infty)} \\ \bar{r}_{pcu} &= \tfrac{r_{pcu}}{(1+\|y\|_\infty)} \\ \bar{r}_{cons} &= \tfrac{r_{cons}}{(1+\|x\|_\infty)} \\ \bar{r}_l &= r_l \\ \bar{r}_u &= r_u. \end{aligned} \tag{5.2}$$

The subproblems are terminated when the following conditions are met:

$$\begin{aligned} \|\bar{r}_L\|_\infty &\leq \alpha_1\mu \\ \|\bar{r}_{pcu}; \bar{r}_{pcl}\|_\infty &\leq \alpha_2\mu \\ \|\bar{r}_{cons}\|_\infty &\leq \alpha_3\mu \\ \|\bar{r}_l; \bar{r}_u\|_\infty &\leq \alpha_4\mu \\ Z^T(\bar{H}+D)Z &\succ -\beta. \end{aligned} \tag{5.3}$$

The default value for α_1 and α_2 is 0.5 and the default value for α_3 and α_4 is 0.1. The default value for β is 10^{-4}. The first four criteria ensure that the normalized residuals have been sufficiently decreased. The last condition ensures that there are no strong directions of negative curvature remaining to be explored. The algorithm is terminated when

$$\mu < \tau \tag{5.4}$$

and the corresponding subproblem converges. The default value of τ is 10^{-5}. The Lagrangian and complementarity equations are purely mathematical constructs. On the other

hand, the constraints often represent real-life bounds on decision variables and conditions. For example, the constraints may represent limitations on some resources, and so the user expects the final solution to satisfy them. Thus, it is desirable to have a final solution that satisfies the bound and constraint equations more accurately than the other equations. It is for this reason that the convergence criteria for the bound and constraint equations is somewhat more stringent than that for the other equations.

5.4 IPSOL Interface to CUTEr Test-set

CUTEr [GOT03] is the de-facto testing environment for production level codes. It is the successor to CUTE (acronym for Constrained and Unconstrained Test Environment), which was created for testing the nonlinear optimizer LANCELOT [CGT92]. CUTEr provides APIs in FORTRAN and MATLAB to enable an optimizer to link easily to a collection of about 1000 test problems. These test problems come from academic research as well as from industrial applications, and their number is continuously growing.

The FORTRAN API has been extensively used to create interfaces for a number of state-of-the-art solvers like SNOPT, MINOS, IPOPT and KNITRO. The MATLAB API of CUTEr, however, is experimental in nature. The current beta release of CUTEr (aptly named CUTEr revisited) fixes quite a few of the bugs of the original CUTEr MATLAB API. Additionally, it also relaxes the memory limitations, which were caused by interfacing MATLAB to FORTRAN 77 code. IPSOL uses the MATLAB API from this beta version of CUTEr to construct an interface for the test problems.

Also, it should be noted that CUTEr and IPSOL use different sign conventions for the dual variables. For IPSOL, the Hessian of the Lagrangian at an iterate (x, y) is evaluated as $\nabla^2\mathscr{L}(x, y) = \nabla^2 f - \sum_i y_i \nabla^2 c_i$, where c_i is a general constraint. However, CUTEr's MATLAB function "sphess" computes it as $\nabla^2\mathscr{L}(x, y) = \nabla^2 f + \sum_i y_i \nabla^2 c_i$. So, the sign of dual variables is reversed when they are passed from IPSOL to CUTEr.

5.5 Performance on CUTEr Small-Scale Problems

To improve the robustness of the algorithm and to identify and fix bugs, IPSOL was first run on a set of 302 small problems from CUTEr. All these problems have less than 100 variables and constraints. The small size of the problems allowed rapid re-runs of the optimizer on the problem set, thereby permitting a quick debugging. We wish to emphasize that even though these problems are small in size, they are numerically quite complex. The chosen

problems include the famous Hock and Schittkowski collection [HS81] and the problems used by Moguerza and Prieto in [MP03]. The results of the final runs are presented in detail in Table 5.1. The names of the columns of Table 5.1 are described below.

- **Problem:** The name of the problem as given in the CUTEr test set. For example, the 13th problem from the Hock and Schittkowski collection is named HS13. The problems where IPSOL fails to converge are marked by a star appended to their name.
- **# var:** Dimension of the primal variable in the original problem. This number is directly obtained by querying CUTEr and does not include the slacks introduced by IPSOL.
- **# constr:** Total number of equality and inequality constraints in the problem. In terms of problem (1.1), $m = m_{\mathscr{E}} + m_{\mathscr{I}}$.
- **# iter:** Total number of iterations to satisfy the convergence criteria. This is the sum total of *minor* iterations across all the subproblems.
- **objective:** Objective value at the final iterate.
- **Lagrangian residual:** Maximum unnormalized residual in the Lagrangian equations, $\|r_L\|_\infty$.
- **infeasibility:** Maximum unnormalized residual in the general constraints, $\|r_{cons}\|_\infty$.

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
1	AIRCRFTA	8	5	2	0.000000e+00	9.9e-09	3.8e-06
2	AIRCRFTB	8	0	10	2.752207e-10	3.7e-06	0.0e+00
3	AIRPORT	84	42	20	4.795270e+04	1.0e-06	4.0e-12
4	ALSOTAME	2	1	11	8.208611e-02	1.3e-07	0.0e+00
5	BIGGSC4	4	7	15	-2.449999e+01	6.2e-07	1.8e-15
6	BT1	2	1	16	-9.999973e-01	2.3e-07	2.7e-08
7	BT10	2	2	8	-1.000000e+00	3.2e-13	1.7e-14
8	BT11	5	3	6	8.248917e-01	6.6e-07	2.9e-08
9	BT12	5	3	4	6.188119e+00	1.9e-09	3.2e-13
10	BT13	5	1	44	9.999558e-07	4.8e-06	4.9e-06
11	BT2	3	1	11	3.256820e-02	8.9e-10	7.8e-10
12	BT3	5	3	1	4.093023e+00	4.0e-15	2.2e-16
13	BT4	3	2	5	-4.551055e+01	7.2e-08	3.9e-08
14	BT5	3	2	7	9.617152e+02	1.7e-10	2.4e-11
15	BT6	5	2	9	2.770448e-01	3.8e-08	2.0e-09
16	BT7	5	3	16	3.065000e+02	3.1e-09	6.3e-13

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
17	BT8	5	2	8	1.000015e+00	6.1e-11	1.5e-05
18	BT9	4	2	15	-1.000000e+00	1.7e-09	9.1e-11
19	CANTILVR	5	1	14	1.339957e+00	2.9e-09	8.2e-10
20	CB2	3	3	12	1.952226e+00	1.1e-07	1.1e-07
21	CB3	3	3	12	2.000003e+00	1.2e-08	2.8e-08
22	CHACONN1	3	3	10	1.952226e+00	1.1e-07	1.1e-07
23	CHACONN2	3	3	12	2.000003e+00	1.2e-08	2.8e-08
24	CONGIGMZ	3	5	19	2.800000e+01	2.2e-06	7.8e-06
25	CSFI1	5	4	20	-4.907520e+01	1.4e-06	1.0e-05
26	CSFI2	5	4	43	5.501761e+01	3.7e-06	2.7e-05
27	DALLASM	196	151	20	-4.819819e+04	7.1e-06	4.0e-13
28	DALLASS	46	31	21	-3.239322e+04	8.6e-06	7.1e-15
29	DECONVB	61	0	306	4.777552e-06	9.2e-06	0.0e+00
30	DECONVC	61	1	37	2.579186e-03	1.5e-06	4.2e-10
31	DEGENLPA	20	15	18	9.379951e-06	1.1e-04	4.0e-16
32	DEGENLPB	20	15	19	-2.138539e+02	1.9e-06	1.3e-15
33	DEMYMALO	3	3	14	-2.999997e+00	2.4e-08	2.3e-09
34	DENSCHNA	2	0	5	2.213698e-12	3.0e-06	0.0e+00
35	DENSCHNB	2	0	13	7.500085e-11	8.6e-11	0.0e+00
36	DENSCHNC	2	0	10	6.253553e-12	1.6e-09	0.0e+00
37	DENSCHND	3	0	36	4.612712e-11	1.1e-07	0.0e+00
38	DENSCHNE	3	0	10	1.173153e-20	2.2e-10	0.0e+00
39	DENSCHNF	2	0	6	3.906514e-13	7.0e-10	0.0e+00
40	DISC2	29	23	35	1.562502e+00	2.3e-10	8.2e-08
41	DIXCHLNG	10	5	9	2.471898e+03	3.4e-08	5.0e-14
42	DNIEPER	61	24	20	1.874401e+04	1.9e-05	2.5e-06
43	DUAL1	85	1	18	3.503789e-02	5.7e-08	4.7e-16
44	DUAL2	96	1	16	3.376385e-02	4.9e-09	0.0e+00
45	DUAL3	111	1	16	1.358550e-01	9.2e-09	3.1e-16
46	DUAL4	75	1	15	7.461191e-01	7.4e-09	3.2e-16
47	DUALC1	9	215	52	6.155251e+03	3.7e-06	6.8e-13
48	DUALC2*	7	229	49	3.551310e+03	6.9e-02	6.8e-13
49	DUALC5	8	278	30	4.272324e+02	7.4e-05	1.7e-13
50	DUALC8	8	503	49	1.830936e+04	2.4e-05	6.8e-13
51	EG1	3	0	13	-1.429306e+00	2.4e-08	0.0e+00
52	EG2	1000	0	3	-9.989474e+02	6.0e-09	0.0e+00
53	EIGMAXA	101	101	18	-9.999990e-01	4.7e-10	1.3e-10
54	EIGMAXB	101	101	14	-9.674354e-04	1.3e-11	2.7e-11
55	EIGMINA	101	101	14	-9.999990e-01	8.8e-11	3.4e-13
56	EIGMINB	101	101	13	4.903541e-01	2.1e-13	4.0e-13
57	ENGVAL2	3	0	16	1.398333e-11	8.2e-06	0.0e+00
58	EXPFITA	5	22	22	1.210894e-03	9.4e-06	2.7e-14
59	EXPFITB	5	102	26	5.131554e-03	3.3e-06	1.9e-14
60	EXTRASIM	2	1	9	1.000001e+00	2.2e-16	2.2e-16
61	FCCU	19	8	11	1.114911e+01	1.5e-09	3.6e-15
62	FLETCHER	4	4	11	1.952537e+01	1.0e-06	2.3e-12
63	GENHS28	10	8	1	9.271737e-01	1.6e-14	1.1e-16
64	GIGOMEZ1	3	3	14	-2.999997e+00	3.3e-08	9.1e-10

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
65	GIGOMEZ2	3	3	12	1.952226e+00	1.1e-07	1.1e-07
66	GIGOMEZ3	3	3	12	2.000003e+00	1.2e-08	2.8e-08
67	GMNCASE1	175	300	13	2.670524e-01	1.7e-08	3.9e-16
68	GMNCASE4	175	350	13	5.946885e+03	1.8e-04	4.7e-09
69	GOFFIN	51	50	9	5.001983e-05	1.5e-14	1.3e-17
70	GOTTFR	2	2	6	0.000000e+00	1.2e-13	3.3e-13
71	GULF	3	0	26	1.849691e-03	4.4e-06	0.0e+00
72	HATFLDA	4	0	9	9.318289e-09	1.1e-08	0.0e+00
73	HATFLDB	4	0	14	5.576135e-03	4.5e-06	0.0e+00
74	HATFLDC	25	0	9	5.547757e-10	1.3e-09	0.0e+00
75	HATFLDD	3	0	29	1.416389e-07	3.6e-07	0.0e+00
76	HATFLDE	3	0	20	5.320425e-07	4.3e-07	0.0e+00
77	HATFLDF*	3	3	37	0.000000e+00	5.7e-09	7.2e-04
78	HATFLDG	25	25	11	0.000000e+00	3.3e-08	3.3e-12
79	HATFLDH	4	7	15	-2.449999e+01	2.0e-08	1.8e-15
80	HELIX	3	0	17	2.540510e-13	1.6e-07	0.0e+00
81	HILBERTA	2	0	1	8.748796e-28	1.3e-14	0.0e+00
82	HILBERTB	10	0	1	2.204630e-18	4.3e-09	0.0e+00
83	HIMMELBB	2	0	15	5.686084e-15	1.2e-06	0.0e+00
84	HIMMELBG	2	0	5	1.199005e-15	8.6e-08	0.0e+00
85	HIMMELBH	2	0	2	-1.000000e+00	3.7e-12	0.0e+00
86	HIMMELBI	100	12	14	-1.735569e+03	2.4e-07	2.2e-14
87	HIMMELBK	24	14	64	2.400881e-05	3.4e-10	4.3e-13
88	HIMMELP1	2	0	11	-5.173785e+01	1.0e-09	0.0e+00
89	HIMMELP2	2	1	15	-8.198042e+00	2.3e-10	2.9e-09
90	HIMMELP3	2	2	12	-8.198039e+00	2.3e-10	4.3e-10
91	HIMMELP4	2	3	16	-8.197662e+00	6.8e-10	2.1e-11
92	HIMMELP5	2	3	49	-2.855716e+01	7.6e-08	3.4e-11
93	HIMMELP6	2	5	17	-8.197560e+00	1.0e-10	4.7e-10
94	HONG	4	1	7	2.257109e+01	1.1e-08	1.1e-16
95	HS1	2	0	28	2.031988e-10	1.1e-06	0.0e+00
96	HS10	2	1	14	-9.999990e-01	1.4e-08	2.1e-08
97	HS100	7	4	29	6.806301e+02	3.2e-08	1.0e-09
98	HS100LNP	7	2	7	6.806301e+02	3.5e-09	9.1e-13
99	HS100MOD	7	4	23	6.972779e+02	3.2e-08	8.3e-11
100	HS101	7	5	56	1.809765e+03	9.4e-06	1.1e-11
101	HS102	7	5	110	9.118806e+02	9.3e-05	6.5e-10
102	HS103	7	5	107	5.436680e+02	5.8e-05	4.3e-10
103	HS104	8	5	15	3.951167e+00	2.6e-07	8.4e-09
104	HS105	8	1	32	1.044612e+03	5.8e-06	4.4e-17
105	HS106	8	6	23	7.050319e+03	6.6e-06	3.8e-10
106	HS107	9	6	10	5.055012e+03	4.5e-05	1.8e-13
107	HS108	9	13	24	-8.660194e-01	1.6e-06	2.3e-09
108	HS109*	9	10	36	5.415828e+03	1.8e-02	2.3e-10
109	HS11	2	1	6	-8.498462e+00	7.6e-07	8.3e-09
110	HS110	10	0	16	-4.577848e+01	6.4e-09	0.0e+00
111	HS111	10	3	12	-4.776111e+01	2.4e-06	1.2e-06
112	HS111LNP	10	3	11	-4.776109e+01	5.4e-08	2.6e-08

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
113	HS112	10	3	15	-4.776109e+01	2.6e-05	6.9e-17
114	HS113	10	8	17	2.430622e+01	2.0e-08	4.0e-08
115	HS114	10	11	39	-1.525877e+03	5.5e-07	6.3e-11
116	HS116	13	14	87	9.758752e+01	8.6e-05	5.0e-14
117	HS117	15	5	16	3.234869e+01	2.9e-07	1.8e-07
118	HS118	15	17	23	6.648205e+02	5.9e-09	5.8e-15
119	HS119	16	8	12	2.448997e+02	2.1e-06	7.5e-16
120	HS12	2	1	13	-3.000000e+01	6.7e-10	4.4e-10
121	HS13	2	1	16	4.341025e-06	1.9e-09	6.2e-13
122	HS14	2	2	9	1.393466e+00	1.2e-07	1.0e-08
123	HS15	2	2	22	3.065000e+02	1.4e-04	2.6e-13
124	HS16	2	2	17	2.500011e-01	2.0e-06	9.8e-09
125	HS17	2	2	15	1.000005e+00	1.0e-08	3.9e-08
126	HS18	2	2	18	5.021487e+00	1.7e-09	6.0e-08
127	HS19	2	2	14	-6.961814e+03	7.1e-07	4.2e-14
128	HS2	2	0	15	4.941230e+00	2.4e-07	0.0e+00
129	HS20	2	3	8	4.019873e+01	1.7e-05	1.4e-12
130	HS21	2	1	13	-9.996000e+01	1.5e-07	1.8e-15
131	HS21MOD	7	1	17	-9.595999e+01	2.7e-09	1.8e-15
132	HS22	2	2	12	1.000002e+00	1.5e-08	9.8e-09
133	HS23	2	5	12	2.000002e+00	1.7e-07	2.5e-08
134	HS24	2	3	10	-9.999981e-01	6.7e-07	8.9e-16
135	HS25	3	0	36	1.845683e-03	7.3e-08	0.0e+00
136	HS26	3	1	16	1.394579e-08	1.1e-07	1.6e-08
137	HS268	5	5	15	1.753361e-05	1.4e-06	3.6e-15
138	HS27	3	1	26	4.000000e-02	2.0e-09	1.5e-09
139	HS28	3	1	1	7.812130e-11	1.2e-14	2.2e-16
140	HS29	3	1	11	-2.262742e+01	1.1e-08	9.7e-11
141	HS3	2	0	9	1.000000e-06	3.3e-12	0.0e+00
142	HS30	3	1	14	1.000002e+00	1.2e-04	8.7e-14
143	HS31	3	1	7	6.000001e+00	4.1e-07	3.0e-10
144	HS32	3	2	14	1.000006e+00	2.0e-10	5.1e-09
145	HS33	3	2	12	-4.585783e+00	1.7e-07	3.9e-08
146	HS34	3	2	22	-8.340293e-01	1.5e-07	1.1e-06
147	HS35	3	1	5	1.111122e-01	2.2e-11	3.3e-17
148	HS35I	3	1	11	1.111122e-01	2.7e-11	4.8e-18
149	HS36	3	1	8	-3.300000e+03	1.2e-05	1.5e-15
150	HS37	3	2	9	-3.456000e+03	1.9e-09	1.8e-15
151	HS38	4	0	56	5.683077e-12	2.8e-07	0.0e+00
152	HS39	4	2	15	-1.000000e+00	1.7e-09	9.1e-11
153	HS3MOD	2	0	9	1.000000e-06	2.2e-12	0.0e+00
154	HS4	2	0	9	2.666669e+00	1.1e-09	0.0e+00
155	HS40	4	3	3	-2.500000e-01	2.7e-09	1.9e-10
156	HS41	4	1	10	1.925927e+00	5.4e-10	0.0e+00
157	HS42	4	2	3	1.385786e+01	3.1e-08	5.2e-09
158	HS43	4	3	11	-4.400000e+01	1.2e-08	4.9e-09
159	HS44	4	6	11	-1.300000e+01	4.0e-07	8.9e-16
160	HS44NEW	4	6	11	-1.300000e+01	4.0e-07	8.9e-16

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
161	HS45	5	0	13	1.000005e+00	7.2e-08	0.0e+00
162	HS46	5	2	16	2.764619e-07	1.7e-07	1.4e-09
163	HS47	5	3	17	1.109930e-08	6.1e-08	1.1e-09
164	HS48	5	2	1	8.874685e-30	8.4e-15	4.4e-16
165	HS49	5	2	16	1.346928e-07	1.2e-07	0.0e+00
166	HS5	2	0	10	-1.913223e+00	1.4e-09	0.0e+00
167	HS50	5	3	8	2.907757e-11	2.2e-06	4.4e-16
168	HS51	5	3	1	1.163514e-12	1.1e-15	0.0e+00
169	HS52	5	3	1	5.326648e+00	3.6e-15	1.4e-17
170	HS53	5	3	6	4.093023e+00	3.9e-10	4.2e-17
171	HS54	6	1	18	-2.420045e-03	7.8e-07	3.6e-12
172	HS55	6	6	10	6.333335e+00	3.3e-08	4.4e-16
173	HS56	7	4	72	-3.456000e+00	2.5e-07	1.5e-08
174	HS57*	2	1	64	9.674067e+01	9.1e-02	5.4e-16
175	HS59	2	3	18	-7.802393e+00	8.5e-11	9.5e-10
176	HS6	2	1	4	2.249843e-10	4.5e-17	4.4e-15
177	HS60	3	1	8	3.256820e-02	2.9e-10	2.0e-11
178	HS61	3	2	6	-1.436461e+02	1.0e-10	1.6e-11
179	HS62	3	1	8	-2.627251e+04	5.0e-08	4.9e-17
180	HS63	3	2	16	9.617152e+02	5.2e-10	1.3e-10
181	HS64	3	1	21	6.299842e+03	1.5e-06	1.5e-13
182	HS65	3	1	18	9.535298e-01	1.5e-09	7.8e-09
183	HS66	3	2	18	5.181653e-01	3.1e-08	1.0e-07
184	HS67	3	14	119	-5.849476e+02	1.9e-06	2.3e-04
185	HS68	4	2	22	-9.204250e-01	3.7e-08	3.4e-10
186	HS69	4	2	15	-9.567129e+02	3.1e-05	1.4e-11
187	HS7	2	1	9	-1.732051e+00	6.7e-12	5.4e-12
188	HS70	4	1	28	7.501784e-03	6.6e-06	1.1e-08
189	HS71	4	2	11	1.701402e+01	8.0e-08	2.5e-07
190	HS72	4	2	17	7.276794e+02	1.6e-05	9.8e-19
191	HS73	4	3	11	2.989438e+01	2.2e-08	2.3e-08
192	HS74	4	5	13	5.126499e+03	1.0e-10	1.5e-12
193	HS75	4	5	14	5.174413e+03	1.1e-10	6.8e-13
194	HS76	4	3	10	-4.681816e+00	5.9e-08	3.0e-16
195	HS76I	4	3	11	-4.681816e+00	5.9e-08	2.1e-16
196	HS77	5	2	8	2.415051e-01	3.4e-08	2.6e-09
197	HS78	5	3	4	-2.919700e+00	1.8e-09	5.6e-12
198	HS79	5	3	4	7.877682e-02	5.3e-09	3.0e-09
199	HS8	2	2	4	-1.000000e+00	3.5e-16	1.4e-11
200	HS80	5	3	8	5.394985e-02	3.3e-09	2.5e-08
201	HS81	5	3	14	5.394984e-02	3.8e-07	2.0e-07
202	HS83	5	3	11	-3.066554e+04	3.3e-05	2.2e-14
203	HS84	5	3	40	-5.280335e+06	7.5e-05	8.7e-11
204	HS85	5	21	74	-1.723368e+00	3.9e-03	3.6e-11
205	HS86	5	10	17	-3.234867e+01	5.7e-07	7.1e-15
206	HS88	2	1	17	1.362658e+00	1.5e-06	3.3e-11
207	HS89*	3	1	20	1.362658e+00	1.5e-06	3.3e-11
208	HS9	2	1	3	-4.999844e-01	8.6e-06	5.7e-14

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
209	HS90	4	1	9	4.896777e-20	3.3e-10	0.0e+00
210	HS91	5	1	46	1.362658e+00	7.5e-07	3.2e-11
211	HS92	6	1	125	1.362658e+00	2.1e-06	2.7e-11
212	HS93	6	2	10	1.350760e+02	2.3e-07	8.8e-11
213	HS95	6	4	25	1.562522e-02	9.6e-06	3.4e-11
214	HS96	6	4	23	1.562514e-02	8.4e-06	1.1e-11
215	HS97	6	4	37	4.071252e+00	2.1e-05	2.0e-07
216	HS98	6	4	50	3.135815e+00	1.2e-04	1.1e-11
217	HS99	7	2	8	-8.310799e+08	1.2e-05	1.3e-10
218	HS99EXP	31	21	17	-1.143818e+12	1.2e-04	4.7e-09
219	HUBFIT	2	1	10	1.689450e-02	1.3e-09	4.1e-18
220	KIWCRESC	3	2	12	2.002198e-06	1.9e-08	4.4e-09
221	LAUNCH	25	28	96	9.005988e+00	1.2e-06	2.7e-08
222	LEAKNET	156	153	37	1.104214e+01	1.7e-08	6.3e-09
223	LIN	4	2	7	-1.757754e-02	1.8e-07	5.6e-17
224	LINSPANH	97	33	45	-7.701000e+01	3.6e-07	7.0e-13
225	LOADBAL	31	31	21	2.059975e+00	4.4e-10	9.2e-13
226	LOGHAIRY	2	0	51	1.823216e-01	9.0e-08	0.0e+00
227	LOGROS	2	0	24	7.640133e-11	2.9e-05	0.0e+00
228	LOOTSMA	3	2	12	1.414217e+00	1.7e-07	3.9e-08
229	LSNNODOC	5	4	11	1.231125e+02	3.6e-09	4.4e-16
230	LSQFIT	2	1	11	3.378799e-02	5.0e-08	2.4e-17
231	MADSEN	3	6	16	6.164342e-01	6.2e-07	3.2e-07
232	MAKELA1	3	2	37	-1.414212e+00	9.9e-09	9.9e-09
233	MAKELA2	3	3	12	7.200027e+00	7.0e-08	2.5e-09
234	MAKELA3	21	20	14	2.000082e-05	1.5e-10	9.7e-11
235	MAKELA4	21	40	9	4.000874e-05	2.8e-13	2.0e-18
236	MANCINO	100	0	9	1.464535e-12	3.3e-07	0.0e+00
237	MARATOS	2	1	3	-1.000000e+00	3.6e-09	1.6e-08
238	MARATOSB	2	0	670	-1.000000e+00	4.0e-06	0.0e+00
239	MATRIX2	6	2	17	4.945822e-06	6.9e-07	2.6e-07
240	MDHOLE	2	0	26	1.000000e-06	6.8e-11	0.0e+00
241	METHANB8	31	31	2	0.000000e+00	1.2e-07	1.0e-07
242	METHANL8	31	31	4	0.000000e+00	1.5e-11	3.7e-11
243	MEXHAT	2	0	27	-4.001000e-02	8.2e-06	0.0e+00
244	MIFFFLIN1	2	0	27	-4.001000e-02	8.2e-06	0.0e+00
245	MIFFFLIN2	2	0	27	-4.001000e-02	8.2e-06	0.0e+00
246	MINMAXBD	5	20	45	1.157066e+02	2.0e-09	9.3e-10
247	MINMAXRB	3	4	12	4.000008e-06	1.9e-10	4.6e-09
248	MISTAKE	9	13	32	-9.999960e-01	9.5e-06	1.2e-08
249	MRIBASIS	36	55	25	1.821790e+01	2.4e-06	4.1e-08
250	MWRIGHT	5	3	8	2.497881e+01	5.2e-07	1.9e-08
251	ODFITS	10	6	15	-2.379912e+03	2.6e-11	5.7e-14
252	OSBORNEA	5	0	20	5.716656e-05	8.8e-06	0.0e+00
253	OSBORNEB	11	0	8	4.013790e-02	6.0e-07	0.0e+00
254	PENTAGON	6	15	22	1.402783e-04	5.8e-07	4.4e-16
255	POLAK1	3	2	11	2.718284e+00	4.6e-08	1.0e-13
256	POLAK2	11	2	9	5.459815e+01	2.4e-12	5.2e-14

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
257	POLAK3	12	10	20	5.933006e+00	1.9e-07	3.5e-07
258	POLAK4	3	3	17	3.975262e-06	2.0e-07	3.2e-07
259	POLAK5	3	2	10	5.000000e+01	1.1e-12	4.9e-15
260	POLAK6	5	4	58	-4.400000e+01	1.6e-08	4.7e-08
261	PRODPL0	60	29	23	5.879014e+01	1.8e-07	1.3e-09
262	PRODPL1	60	29	36	3.573900e+01	2.0e-06	6.8e-15
263	RK23	17	11	45	8.333826e-02	3.5e-08	9.8e-09
264	ROSENMMX	5	4	28	-4.400000e+01	1.6e-08	4.7e-08
265	S277-280	4	4	10	5.076196e+00	1.2e-08	5.6e-17
266	S308	2	0	8	7.731991e-01	4.2e-06	0.0e+00
267	S316-322	2	1	7	3.343146e+02	4.7e-11	1.5e-13
268	S368	8	0	19	-9.374971e-01	5.2e-08	0.0e+00
269	SWOPF	83	92	31	6.790999e-02	1.3e-08	1.6e-07
270	SYNTHES1	6	6	12	7.592894e-01	1.5e-08	7.4e-10
271	SYNTHES2	11	14	11	-5.543923e-01	6.3e-07	1.6e-08
272	SYNTHES3	17	23	13	1.508220e+01	3.8e-07	5.2e-09
273	TAME	2	1	4	0.000000e+00	2.2e-16	1.1e-16
274	TENBARS1	18	9	32	2.302549e+03	4.3e-07	2.7e-10
275	TENBARS2	18	8	31	2.302549e+03	6.7e-07	2.3e-10
276	TENBARS3	18	8	34	2.247129e+03	1.3e-06	1.3e-09
277	TENBARS4	18	9	35	3.697074e+02	2.6e-08	6.4e-08
278	TFI1	3	101	58	5.334689e+00	6.2e-09	9.2e-10
279	TFI2	3	101	16	6.490339e-01	2.4e-06	2.7e-16
280	TFI3	3	101	22	4.301161e+00	1.3e-06	1.5e-16
281	TRIGGER	7	6	19	0.000000e+00	1.2e-08	1.3e-13
282	TRIMLOSS	142	75	26	9.083943e+00	1.0e-08	4.1e-07
283	TRUSPYR1	11	4	18	1.122875e+01	8.5e-07	3.0e-11
284	TRUSPYR2	11	11	23	1.122875e+01	5.8e-05	1.3e-14
285	TRY-B	2	1	14	2.022836e-11	5.4e-10	2.4e-09
286	TWOBARS	2	2	11	1.508653e+00	1.9e-08	4.3e-09
287	WEEDS	3	0	39	2.587725e+00	1.4e-05	0.0e+00
288	WOMFLET	3	3	21	6.050001e+00	1.1e-08	3.3e-09
289	YFIT	3	0	52	5.783854e-03	1.8e-05	0.0e+00
290	YFITU	3	0	34	5.783396e-03	2.6e-05	0.0e+00
291	ZAMB2-10	270	96	24	-1.390128e+00	1.7e-08	9.7e-06
292	ZAMB2-11	270	96	20	-1.439861e+00	3.6e-08	1.3e-05
293	ZAMB2-8	138	48	26	-1.383525e-01	1.6e-08	1.1e-06
294	ZAMB2-9	138	48	23	-3.315925e-01	1.6e-09	2.4e-06
295	ZANGWIL2	2	0	1	-1.820000e+01	1.1e-11	0.0e+00
296	ZANGWIL3	3	3	1	0.000000e+00	4.1e-20	2.0e-14
297	ZECEVIC2	2	2	8	-4.124999e+00	2.6e-09	3.0e-17
298	ZECEVIC3	2	2	7	9.730945e+01	7.3e-10	7.1e-11
299	ZECEVIC4	2	2	10	7.557509e+00	6.4e-10	9.0e-11
300	ZY2	3	2	11	5.615102e+00	2.1e-08	2.0e-08

Table 5.1: Performance of IPSOL on 300 small problems from the CUTEr testset.

With the default parameters, IPSOL is able to solve 295 out of the 302 problems. HS57 solves with a larger starting value of the barrier parameter. However, this has not been used as the aim here is to present the robustness and performance of IPSOL with its default parameters.

The causes of failure for some of the problems are noted here.

- Severely ill-conditioned Hessian: HS109, HS85.
- Domain range error while evaluating the functions: DECONVNE (not listed).
- Non-smooth problem: HS87 (not listed).

5.6 Performance on a Few CUTEr Large-Scale Problems

IPSOL has been designed and carefully implemented to be used for solving large-scale optimization problems. However the memory limitation of MATLAB did limit the efforts to test IPSOL on large problems. Table 5.2 contains the performance of IPSOL on 42 large-scale problems. It should be noted that the average number of iterations is almost the same as for the smaller test set.

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
1	BRATU3D	4913	3375	3	0.000000e+00	5.6e-12	3.0e-09
2	CAMSHAPE	800	1603	58	-4.273018e+00	9.2e-06	3.3e-10
3	CATENA	3003	1000	25	-2.099583e+06	3.9e-05	2.4e-13
4	DALLASL	906	667	17	-2.026041e+05	1.8e-04	9.0e-13
5	DEGEN2	534	444	25	-3.944930e+03	1.4e-05	1.7e-14
6	DTOC1NA	5998	3996	8	4.138867e+00	3.7e-07	2.1e-10
7	DTOC1NB	5998	3996	9	7.138849e+00	5.2e-09	2.8e-11
8	DTOC1NC	5998	3996	5	3.519934e+01	1.5e-06	1.7e-08
9	DTOC1ND	5998	3996	6	4.760303e+01	1.5e-07	4.1e-09
10	EXPFITC	5	502	27	2.394353e-02	6.4e-07	3.0e-14
11	EXPLIN	1200	0	35	-7.192548e+07	3.8e-05	0.0e+00
12	EXPLIN2	1200	0	24	-7.199883e+07	4.8e-07	0.0e+00
13	EXPQUAD	1200	0	49	-3.684941e+09	4.5e-07	0.0e+00
14	GMNCASE2	175	1050	12	-9.943578e-01	2.9e-08	8.9e-16
15	GMNCASE3	175	1050	11	1.525230e+00	6.9e-08	4.4e-16
16	HAGER1	5001	2500	1	8.808343e-01	1.7e-12	2.6e-11
17	HAGER2	5001	2500	1	4.321334e-01	6.3e-12	1.8e-11
18	HAGER3	5001	2500	1	1.409828e-01	1.4e-12	2.1e-11
19	HAGER4	5001	2500	10	2.798288e+00	1.0e-11	4.8e-12
20	HELSBY	1408	1399	41	3.462518e+01	5.4e-06	7.6e-07
21	HET-Z	2	1002	16	1.000000e+00	5.7e-09	4.4e-16
22	LISWET1	2002	2000	26	7.223894e+00	4.3e-06	5.1e-16
23	LISWET10	2002	2000	30	9.898246e+00	7.8e-07	4.0e-16

	Problem	# var	# constr	# iter	objective	Lagrangian residual	infeasibility
24	LISWET2	2002	2000	43	4.999258e+00	1.4e-05	2.7e-16
25	LISWET3	2002	2000	29	4.998007e+00	1.0e-06	2.4e-16
26	LISWET4	2002	2000	30	4.998130e+00	8.2e-07	3.7e-16
27	LISWET5	2002	2000	31	4.998060e+00	1.8e-06	8.0e-16
28	LISWET6	2002	2000	26	4.998110e+00	3.1e-06	2.0e-16
29	LISWET7	2002	2000	26	9.989716e+01	7.9e-05	2.1e-16
30	LISWET8	2002	2000	31	1.431324e+02	3.0e-06	3.8e-16
31	LISWET9	2002	2000	30	3.929222e+02	1.9e-05	1.8e-16
32	OET1	3	1002	17	5.382471e-01	1.4e-08	6.8e-16
33	OET2	3	1002	25	8.716620e-02	1.6e-07	1.7e-11
34	OET3	4	1002	13	7.185322e-03	6.3e-08	2.2e-15
35	OET4	4	1002	17	7.040527e-03	3.9e-06	6.7e-08
36	OET5	5	1002	36	2.838259e-03	4.5e-08	3.2e-09
37	OET6	5	1002	62	8.774781e-01	1.2e-03	4.9e-02
38	PENALTY1	1000	0	39	9.686178e-03	1.1e-06	0.0e+00
39	PROBPENL	500	0	6	3.974522e-07	2.3e-07	0.0e+00
40	READING1	4002	2000	54	-1.061640e-01	7.0e-03	4.8e-03
41	READING2	6003	4000	4	-4.454431e-03	2.7e-07	1.6e-13
42	READING3	4002	2001	31	-8.066820e-02	8.3e-03	4.0e-03

Table 5.2: Performance of IPSOL on 42 large problems from the CUTEr testset.

5.7 Comparisons with IPOPT and LOQO

In this section, the performance of IPOPT is compared with the well known non-convex interior point solvers IPOPT and LOQO. These solvers have been in active use for over a decade now and are continuously being improved. The number of iterations of IPOPT on the small CUTEr test set (section 5.5) is compared with the number of iterations of the other two solvers. The performance of the other two solvers was taken from [WB06]. The termination tolerance for LOQO was set to 10^{-6} while for IPOPT it was set to 10^{-8}. It should be noted that a direct comparison of the termination tolerances across various solvers is not possible because of the different scaling involved in normalizing the residual. However, these termination tolerances are in the ballpark of the termination tolerance chosen for IPSOL. The comparisons are presented as histograms (Figure 5.1 and 5.2). The performance of IPSOL is more often better than LOQO, but less often as good as IPOPT. We believe this difference is largely due to the initialization and reduction strategies of the barrier parameter μ [NWW05].

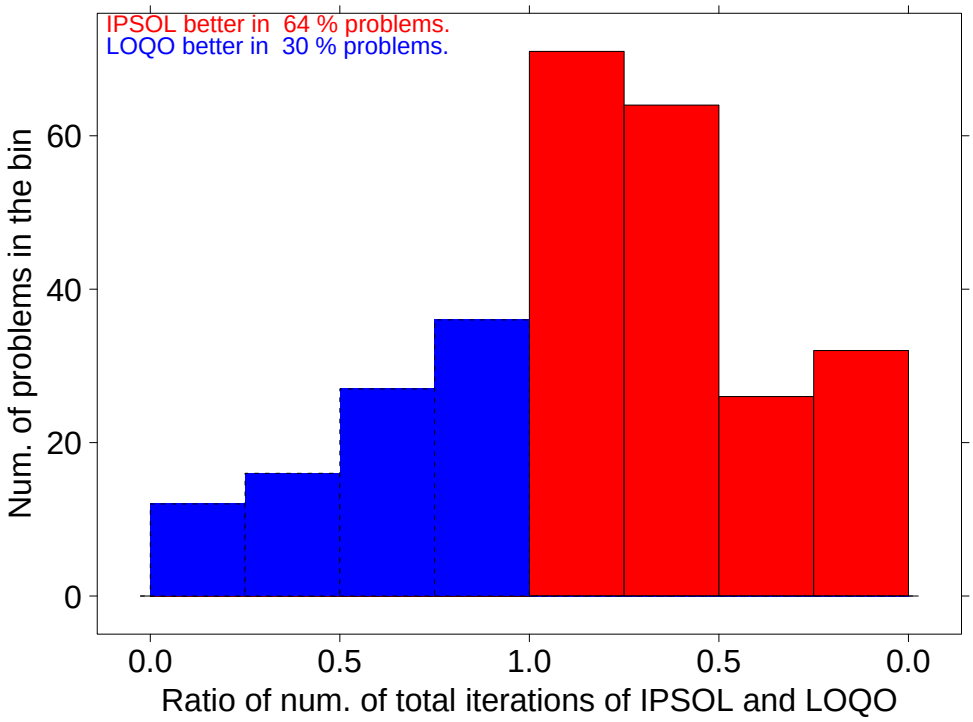

Figure 5.1: The red histogram (on the right) is for the ratio of the number of iterations of IPSOL to the number of iterations of LOQO. This histogram includes only those problems from Table 5.1 for which IPSOL has a strictly lower iteration count than IPOPT. The blue histogram is its counterpart for the problems where LOQO has a lower iteration count than IPSOL.

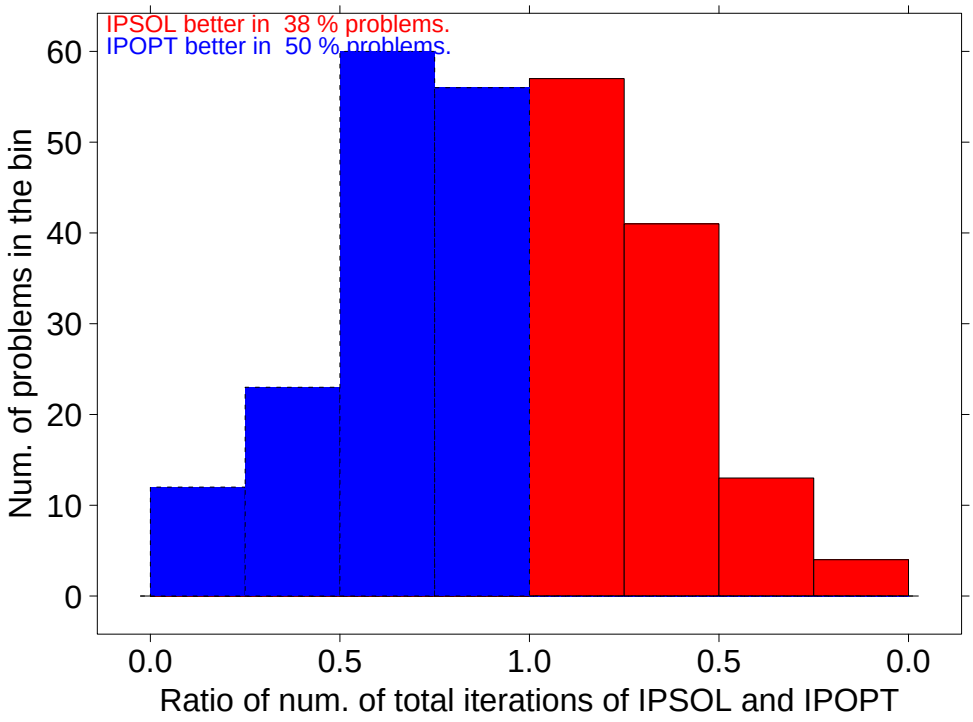

Figure 5.2: The red histogram (on the right) is for the ratio of the number of iterations of IPSOL to the number of iterations of IPOPT. This histogram includes only those problems from Table 5.1 for which IPSOL has a strictly lower iteration count than IPOPT. The blue histogram is its counterpart for the problems where IPOPT has a lower iteration count than IPSOL.

Chapter 6

Contributions and Future Research

In this thesis we proposed IPSOL, a method for solving large nonlinearly constrained optimization problems. The method utilizes second derivatives and is based on the theoretical framework of Murray and Prieto [MP95] to solve the subproblems (2.2). Their method has the properties of global convergence and a quadratic rate of convergence. The proofs are based on computing at each iteration a feasible step, a descent direction, and a direction of negative curvature. How to compute such directions is not specified. Here we show one way this may be done. A key step is to remove inequality constraints by using a barrier formulation. Since the barrier subproblems have only equality constraints, the utilization of second derivatives is considerably simplified, but still problematic. The key issue is to identify when the linearized KKT conditions need to be modified (because they point to a saddle point or a maximizer) and how to make a modification. Two other important issues are addressed. The first is how to compute the required directions for large problems for which the degrees of freedom may or may not be large. The second is how to regularize the definition of these directions to cope with ill-conditioning in both the Jacobian of the constraints and the reduced Hessian of the Lagrangian. While the barrier transformation simplifies the computation of the required directions it exacerbates the condition issue. A modification of the Lanczos algorithm is used to compute both the descent direction and the direction of negative curvature.

The algorithm IPSOL has been prototyped in MATLAB and has been tested on a subset of CUTEr problems. While the optimizer is able to solve a large number of problems, it fails to converge on a few cases. We briefly state the reasons of failure of IPSOL on a few CUTEr problems and suggest possible solutions, which may be incorporated in a future version of the solver to make it more robust.

- IPSOL uses a modification of the CG algorithm to compute search directions from the reduced Hessian system (section 3.3.3). This algorithm, like any iterative algorithm, has difficulty in solving severely ill-conditioned systems.
  - Symptom: Merit function has a near zero derivative ($\sim -10^{-12}$) at intermediate iterates.
  - Example cases: HS109.
  - Proposed solution: Keep a direct solver like the modified Cholesky factorization as an option for severely ill-conditioned problems.
- In a few cases, the default choice of the initial point (section 2.5) is not good with respect to the scale of the problem. As a consequence, the initial search direction typically has one or more disproportionately large components. This direction may result in a very small $\alpha_{\max}$, thereby hindering progress towards the solution.
  - Symptom: Maximum permissible step is very small ($\sim 10^{-3}$).
  - Example case: ROBOTARM.
  - Proposed solution: Construct an algorithm to artificially scale down any disproportionally large component(s). The challenge is to be able to do this without affecting the overall convergence of the algorithm.
- An optimization problem may have functions whose domain of definition is restricted (say $\sin^{-1}(\overline{x})$). A user may express this restriction on the domain by placing bounds on $\overline{x}$. However, IPSOL is an infeasible interior point method and does not restrict $\overline{x}$ to respect bounds at intermediate iterates.
  - Symptom: Linesearch evaluation at trial point results in NAN/INFs.
  - Example case: DECONVNE.
  - Proposed solution: The user should be given an option to indicate whether the functions involved in the optimization problem are defined on a restricted domain. In this case, the user should also explicitly bound the variables in this domain. IPSOL should respect such bounds by using an alternative log barrier formulation wherein the logarithms are directly put on $\overline{x} - l$ or $u - \overline{x}$ rather than on the slack variables $t_{l,u}$.

Finally, a more dynamic strategy for initializing and updating μ is worth exploring, as it possibly holds the key to improving the performance of the solver.

Appendix A

Overcoming the Temporary Incompatibility

In the presence of rank deficiency in the Jacobian, the system (2.6) may be incompatible. It is possible that this incompatibility is temporary and the problem is feasible and has a solution. Variables are added to the optimization problems to overcome this temporary incompatibility of the system. The addition of these variables has the effect of relaxing the constraints, thereby helping the solver overcome temporary incompatibility.

A.1 L1 Elastic Variables

In this approach, when rank deficiency is detected in the Jacobian (section 3.5), the objective function is modified by the addition of an L1 penalty for the infeasibilities. At such an iterate, the following optimization is (conceptually) solved:

$$\begin{array}{lllll} \underset{\overline{x},v,w}{\text{minimize}} & f(\overline{x}) + \gamma e^T(v+w) & & \\ \text{s.t.} & c(\overline{x}) + v - w & = & 0 \\ & E_l\overline{x} & \geq & l \\ & E_u\overline{x} & \leq & u \\ & v, w & \geq & 0 \end{array} . \tag{A.1}$$

The addition of of variables t_l, t_u and barrier terms leads to the problem

$$\begin{array}{lllll}
\underset{\overline{x},t_l,t_u,v,w}{\text{minimize}} & f(\overline{x}) - \mu[e_l^T \ln(t_l) + e_u^T \ln(t_u) + e_v^T \ln(t_v) + e_w^T \ln(t_w)] & +\gamma(e_v^T v + e_w^T w) & & \\
\text{s.t.} & c(\overline{x}) + v - w & = & 0 & : y \\
& E_l \overline{x} - t_l & = & l & : z_l \\
& -E_u \overline{x} - t_u & = & -u & : z_u \\
& v - t_v & = & 0 & : z_v \\
& -w - t_w & = & 0 & : z_w,
\end{array} \tag{A.2}$$

whose optimality conditions are

$$\begin{array}{rrrrrrrcrl}
-\nabla f(\overline{x}) & & +\nabla c(\overline{x})\, y & +E_l^T z_l & -E_u^T z_u & & & = & 0 & (L_x) \\
& & y - \gamma e & & & +z_v & & = & 0 & (L_v) \\
& & -y - \gamma e & & & & -z_w & = & 0 & (L_w) \\
& & & T_l Z_l e_l & & & & = & \mu e_l & (pcl) \\
& & & & T_u Z_u e_u & & & = & \mu e_u & (pcu) \\
& & & & & T_v Z_v e_v & & = & \mu e_v & (pcv) \\
& & & & & & T_w Z_w e_w & = & \mu e_w & (pcw) \\
c(\overline{x}) & +v & -w & & & & & = & 0 & (cons) \\
E_l \overline{x} & & & -t_l & & & & = & l & (low) \\
-E_u \overline{x} & & & & -t_u & & & = & -u & (upp) \\
& v & & & & -t_v & & = & 0 & (v) \\
& & -w & & & & -t_w & = & 0 & (w).
\end{array} \tag{A.3}$$

The application of Newton's method to these nonlinear equations leads to the following linearized equations:

$$\begin{bmatrix}
-\nabla^2\mathscr{L} & & & & & & \nabla c(\overline{x}) & E_l^T & -E_u^T & & & \\
& I & & & & & I & & & & & \\
& & -I & & & & -I & & & & & \\
& & & Z_l & & & & T_l & & & & \\
& & & & Z_u & & & & T_u & & & \\
& & & & & Z_v & & & & T_v & & \\
& & & & & & Z_w & & & & T_w & \\
\nabla c(\overline{x})^T & I & -I & & & & & & & & & \\
E_l & & & & & & & & & & & \\
-E_u & & & & & & & & & & & \\
& I & & & & & & & & -I & & \\
& & -I & & & & & & & & -I &
\end{bmatrix}
\begin{bmatrix} \delta\overline{x} \\ \delta v \\ \delta w \\ \delta t_l \\ \delta t_u \\ \delta t_v \\ \delta t_w \\ \delta y \\ \delta z_l \\ \delta z_u \\ \delta z_v \\ \delta z_w \end{bmatrix}
=
\begin{bmatrix} r_{L_x} \\ r_{L_v} \\ r_{L_w} \\ r_{pcl} \\ r_{pcu} \\ r_{pcv} \\ r_{pcw} \\ r_c \\ r_l \\ r_u \\ r_v \\ r_w \end{bmatrix}. \tag{A.4}$$

We can eliminate $\delta z_l, \delta z_u, \delta z_v, \delta z_w, \delta t_l, \delta t_u, \delta t_v, \delta t_w$ (in that order) to obtain a much smaller system

$$\begin{bmatrix}
-\mathscr{H} & & & \nabla c(\overline{x}) \\
& -T_v^{-1}Z_v & & I \\
& & -T_w^{-1}Z_w & -I \\
\nabla c(\overline{x})^T & I & -I &
\end{bmatrix}
\begin{bmatrix} \delta\overline{x} \\ \delta v \\ \delta w \\ \delta y \end{bmatrix}
=
\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_c \end{bmatrix}, \tag{A.5}$$

where

$$\begin{aligned}
\mathscr{H} &= \nabla^2\mathscr{L} + E_l^T T_l^{-1} Z_l E_l + E_u^T T_u^{-1} Z_u E_u \\
r_1 &= r_{L_x} - E_l^T T_l^{-1}(r_{pcl} + Z_l r_l) + E_u^T T_u^{-1}(r_{pcu} + Z_u r_u) \\
r_2 &= r_{L_v} - T_v^{-1}(r_{pcv} + Z_v r_v) \\
r_3 &= r_{L_w} + T_w^{-1}(r_{pcw} + Z_w r_w).
\end{aligned} \tag{A.6}$$

The elastic variables need to be initialized to a positive value. This problem of initializing elastic variables is similar to that of restarting an interior point method. An improper initialization can lead to a slow rate of convergence. As of this writing, restarting a nonlinear interior point method is not well understood and so we do not use introduce the elastic variable midway through the solution. A conservative but computationally expensive alternative is to let the elastic variables always be present in the problem. In this case they become part of the problem definition and are easily incorporated in IPSOL.

Bibliography

[Arm66] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16:1–3, 1966.

[BNW06] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. KNITRO: An integrated package for nonlinear optimization. In *Large-scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 35–59. Springer, New York, 2006.

[BT95] Paul T. Boggs and Jon W. Tolle. Sequential quadratic programming. In *Acta Numerica, 1995*, Acta Numer., pages 1–51. Cambridge Univ. Press, Cambridge, 1995.

[Car61] Charles W. Carroll. The created response surface technique for optimizing nonlinear, restrained systems. *Operations Res.*, 9:169–185, 1961.

[CD00] Tzu-Yi Chen and James W. Demmel. Balancing sparse matrices for computing eigenvalues. In *Proceedings of the International Workshop on Accurate Solution of Eigenvalue Problems (University Park, PA, 1998)*, volume 309, pages 261–287, 2000.

[CGT92] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *LANCELOT: A Fortran package for large-scale nonlinear optimization (release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1992.

[CGT00] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-region Methods*. MPS/SIAM Series on Optimization. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000.

[DNT08] Antoine Deza, Eissa Nematollahi, and Tamas Terlaky. How good are interior point methods? Klee-Minty cubes tighten iteration-complexity bounds. *Math. Program.*, 113(1):1–14, 2008.

[Dru85] Arne Drud. CONOPT: a GRG code for large sparse dynamic nonlinear optimization problems. *Math. Programming*, 31(2):153–191, 1985.

[Eld92] Samuel K. Eldersveld. *Large-Scale Sequential Quadratic Programming Algorithms*. PhD thesis, Stanford University, 1992.

[FGL+02] Roger Fletcher, Nicholas I. M. Gould, Sven Leyffer, Philippe L. Toint, and Andreas Wächter. Global convergence of a trust-region SQP-filter algorithm for general nonlinear programming. *SIAM J. Optim.*, 13(3):635–659 (electronic) (2003), 2002.

[FL02] Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Math. Program.*, 91(2, Ser. A):239–269, 2002.

[FM90] Anthony V. Fiacco and Garth P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, volume 4 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 1990.

[FM93] Anders Forsgren and Walter Murray. Newton methods for large-scale linear equality-constrained minimization. *SIAM J. Matrix Anal. Appl.*, 14(2):560–587, 1993.

[Fri54] K. R. Frisch. Principles of linear programming - with particular reference to the double gradient form of the logarithmic potential method. Technical report, University Institute of Economics, Oslo, Norway, October 1954.

[Fri55] K. R. Frisch. The logarithmic potential method of convex programming. Technical report, University Institute of Economics, Oslo, Norway, May 1955.

[FS05] Michael P. Friedlander and Michael A. Saunders. A globally convergent linearly constrained Lagrangian method for nonlinear optimization. *SIAM J. Optim.*, 15(3):863–897 (electronic), 2005.

[GMPS95] Philip E. Gill, Walter Murray, Dulce B. Ponceleón, and Michael A. Saunders. Primal-dual methods for linear programming. *Math. Programming*, 70(3, Ser. A):251–277, 1995.

[GMS+86] Philip E. Gill, Walter Murray, Michael A. Saunders, J. A. Tomlin, and Margaret H. Wright. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Math. Programming*, 36(2):183–209, 1986.

[GMS02] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM J. Optim.*, 12(4):979–1006 (electronic), 2002.

[GMS05] Philip E. Gill, Walter Murray, and Michael A. Saunders. SNOPT: an SQP algorithm for large-scale constrained optimization. *SIAM Rev.*, 47(1):99–131 (electronic), 2005.

[GMSW79] Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright. Two steplength algorithms for numerical optimization. Technical Report SOL 79-25, Systems Optimization Laboratory, Stanford University, 1979.

[GMW87] Philip E. Gill, Walter Murray, and Margaret H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra Appl.*, 88/89:239–270, 1987.

[GNP97] John R. Gilbert, Esmond G. Ng, and Barry W. Peyton. Separators and structure prediction in sparse orthogonal factorization. *Linear Algebra Appl.*, 262:83–97, 1997.

[Gon97] Jacek Gondzio. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS J. Comput.*, 9(1):73–91, 1997.

[GOT03] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. CUTEr and SifDec: A constrained and unconstrained testing environment, revisited. *ACM Trans. Math. Software*, 29(4):373–394, 2003.

[GT04] Nick Gould and Philippe L. Toint. Preprocessing for quadratic programming. *Math. Program.*, 100(1, Ser. B):95–132, 2004.

[HM79] Shih P. Han and Olvi L. Mangasarian. Exact penalty functions in nonlinear programming. *Math. Programming*, 17(3):251–269, 1979.

[HS52] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *J. Research Nat. Bur. Standards*, 49:409–436 (1953), 1952.

[HS81] Willi Hock and Klaus Schittkowski. *Test examples for nonlinear programming codes*, volume 187 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 1981.

[Kar84] Narendra K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[KM72] Victor Klee and George J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[LFR74] Leon S. Lasdon, Richard L. Fox, and Margery W. Ratner. Nonlinear optimization using the generalized reduced gradient method. *Rev. Française Automat. Informat. Recherche Opérationnelle Sér. Verte*, 8(V-3):73–103, 1974.

[Loo69] F. A. Lootsma. Hessian matrices of penalty functions for solving constrained-optimization problems. *Philips Res. Rep.*, 24:322–330, 1969.

[LPR96] Zhi-Quan Luo, Jong-Shi Pang, and Daniel Ralph. *Mathematical Programs with Equilibrium Constraints*. Cambridge University Press, Cambridge, 1996.

[Meg89] Nimrod Megiddo. Pathways to the optimal set in linear programming. *Progress in Mathemtical Progrmming*, pages 131–158, 1989.

[Meh92] Sanjay Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optim.*, 2(4):575–601, 1992.

[MM79] D. Q. Mayne and N. Maratos. A first order, exact penalty function algorithm for equality constrained optimization problems. *Math. Programming*, 16(3):303–324, 1979.

[MP95] Walter Murray and Francisco J. Prieto. A second-derivative method for nonlinearly constrained optimization. Technical Report SOL 95-3, Systems Optimization Laboratory, Stanford University, 1995.

[MP03] Javier M. Moguerza and Francisco J. Prieto. An augmented Lagrangian interior-point method using directions of negative curvature. *Math. Program.*, 95(3, Ser. A):573–616, 2003.

[MS78] Bruce A. Murtagh and Michael A. Saunders. Large-Scale linearly constrained optimization. *Math. Programming*, 14(1):41–72, 1978.

[MS82] Bruce A. Murtagh and Michael A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. *Math. Programming Stud.*, (16):84–117, 1982. Algorithms for constrained minimization of smooth nonlinear functions.

[MS06] Gérard Meurant and Zdeněk Strakoš. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numer.*, 15:1–71, 2006.

[MT94] Jorge J. Moré and David J. Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Trans. Math. Software*, 20(3):286–307, 1994.

[Mur69] Walter Murray. An algorithm for constrained minimization. *Optimization*, pages 247–258, 1969.

[Mur71] Walter Murray. Analytical expressions for the eigenvalues and eigenvectors of the Hessian matrices of barrier and penalty functions. *J. Optimization Theory Appl.*, 7:189–196, 1971.

[Mur74] Frederic H. Murphy. A class of exponential penalty functions. *SIAM J. Control*, 12:679–687, 1974.

[MW94] Walter Murray and Margaret H. Wright. Line search procedures for the logarithmic barrier function. *SIAM J. Optim.*, 4(2):229–246, 1994.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13 of *SIAM Studies in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1994.

[NW06] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition, 2006.

[NWW05] Jorge Nocedal, Andreas Wächter, and Richard A. Waltz. Adaptive barrier strategies for nonlinear interior methods. Technical Report RC 23563, IBM T.J. Watson Research Center, Yorktown, USA, 2005.

[OR70] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.

[Pie69] Tomasz Pietrzykowski. An exact potential method for constrained maxima. *SIAM J. Numer. Anal.*, 6:299–304, 1969.

[PS82] Christopher C. Paige and Michael A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, 1982.

[Rob72] Stephen M. Robinson. A quadratically-convergent algorithm for general nonlinear programming problems. *Math. Programming*, 3:145–156, 1972.

[SL99] X. L. Sun and D. Li. Logarithmic-exponential penalty formulation for integer programming. *Appl. Math. Lett.*, 12(7):73–77, 1999.

[Sto75] David R. Stoutemyer. Analytical optimization using computer algebraic manipulation. *ACM Trans. Math. Software*, 1:147–164, 1975.

[Tap80] Richard A. Tapia. On the role of slack variables in quasi-Newton methods for constrained optimization. In *Numerical Optimisation of Dynamic Systems*, pages 235–246. North-Holland, Amsterdam, 1980.

[Van99] Robert J. Vanderbei. LOQO: an interior point code for quadratic programming. *Optim. Methods Softw.*, 11/12(1-4):451–484, 1999. Interior point methods.

[WB06] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.*, 106(1, Ser. A):25–57, 2006.

[Wol69] Philip Wolfe. Convergence conditions for ascent methods. *SIAM Rev.*, 11:226–235, 1969.

[Wol71] Philip Wolfe. Convergence conditions for ascent methods. II. Some corrections. *SIAM Rev.*, 13:185–188, 1971.

[Wri05] Margaret H. Wright. The interior-point revolution in optimization: history, recent developments, and lasting consequences. *Bull. Amer. Math. Soc. (N.S.)*, 42(1):39–56 (electronic), 2005.