Linköping Studies in Science and Technology.
Theses No. 689

Sparse Least Squares Problems with Box Constraints

Mikael Adlers

Division of Numerical Analysis
Department of Mathematics
Linköpings universitet, S-581 83 Linköping, Sweden

ISBN 91-7219-216-X
ISSN 0280-7971
LiU-TEK-LIC-1998:24
Printed in Sweden by UniTryck, Linköping 1998

Abstract

In this thesis the sparse least squares problem with box constraints is considered. This problem has the form $\min_{l \leq x \leq u} ||Ax - b||_2$, where some lower/upper bounds may not be present. This type of problem arises from, for example, reconstruction problems in geodesy and tomography. Here methods based on direct factorization methods for sparse matrix computation are explored. Two completely different approaches for solving the problem are discussed and compared, i.e. active set methods and primal-dual interior-point methods. An active set block method suitable for sparse problems is developed and a convergence proof is presented. The interior-point methods compared are all based on Mehrotra's predictor-corrector path following method. Different schemes for choosing the barrier parameter μ and for making multiple corrections are discussed. The problem of solving the seemingly increasingly ill-conditioned subproblems when the interior-point method converges is briefly analyzed.

Numerical comparison of different active set methods and interior-point methods are given. The "best" method from each category are compared against each other. A comparison of the behavior when the solution has different number of free variables is also given. The numerical tests show that the block active set method is faster and gives better accuracy for both nondegenerate and degenerate problems.

Acknowledgements

First of all I would like to express my gratitude to my supervisor Prof. Åke Björck who introduced me to the world of sparse linear algebra and least squares problems. Thank you for giving me an interesting problem with both numerical and theoretical aspects to tackle. I would also like to thank Ph.D. Pontus Matstoms for supporting me and answering my questions about sparse QR and sparse orderings.

Further I would like to thank my friends and colleagues at the Department of Mathematics, in particular the people at the Numerical Analysis group for nice coffee breaks and interesting lectures.

Linköping, May 1998
Mikael Adlers

Contents

1 Introduction

Many scientific modeling applications give rise to the Least Squares Problem. In this thesis we will compare different ways to solve the Least Squares Problem with simple constraints on each variable, the Bounded Least Squares (BLS) Problem,

$$\min_{x\in\Omega} ||Ax-b||_2, \tag{1.1}$$

$$\Omega = \{x \mid l_i \leq x_i \leq u_i,\ \forall i\},$$

where $-\infty \leq l_i,\ u_i \leq \infty$, $A \in I\!R^{m\times n}$ is a sparse matrix and $b \in I\!R^m$. This problem arises in different applications, e.g., from reconstruction problems in geodesy and tomography, ocean circulation models, and construction of optical mirrors. We will look at two different approaches to solving this optimization problem, both using direct methods in the solution process. One of the methods can be used to solve the bounded least squares system with almost the same amount of work as solving the unconstrained least squares problem if no constraints are active. This means that safety bounds can be added to the problem with almost no extra computational cost. A similar comparison of methods for the nonnegative least squares problem was made Portugal, Judice and Vincente [62].

The problem (1.1) can be reformulated as a convex quadratic programming problem (QP) and then solved with one of several QP algorithms, see Coleman and Hulbert [9], Monteiro and Adler [53] or the book by Fletcher [17]. However, if the BLS is reformulated this way by forming the quadratic function we form the matrix A^TA. This can lead to numerical difficulties if the matrix A is ill-conditioned, see [5]. In the algorithms presented here this can be avoided.

1.1 Outline of Thesis

Here in the first section we give a brief introduction to constrained optimization. We state the general first order optimality conditions, the KKT-conditions. Further we show that the least squares problem with box constrains can be formulated as a convex quadratic programming problem. Some of the most common software packages for quadratic programming problems are listed.

In the second section we introduce the reader to the linear least squares problem. The basic methods for solving the least squares problem are discussed together with the standard error analysis. Methods for computing solution of a consistent underdetermined linear system of equations are also discussed. Iterative refinement for both these problems are also described in this section.

In the third section techniques for sparse matrix computation are briefly surveyed. Different storage schemes are discussed together with different orderings of a matrix. A method for storing an implicit representation of Q from a multi-frontal QR factorization is developed. Some numerical experiments demonstrate the efficiency of this approach.

In the fourth section the generation of test problems are presented together with an algorithm for computing a solution with predetermined attributes. The set of sparse test matrices consists of three different subsets. The first subset is matrices from the Harwell-Boeing collection, the second subset of matrices is from animal breeding science and the final subset of matrices is from a finite element model problem. All the test problems have been used in the literature as test problems for sparse least squares methods. In generating the test problems it is important to be able to compute the solution to an underdetermined linear system to high accuracy. Therefore iterative refinement with residuals in quadruple precision, is used. Two different types of test problems, type A and type B are presented. Type A problems are nondegenerated and type B problems have a degenerated solution.

The fifth and sixth sections contain the main issue of this thesis. Here two different approaches for solving the least squares problem with box constraints are studied. Section 5 covers so called active set methods. These methods solve smaller unconstrained least squares problems in each iteration and at the end the index set of the variables bounded at the solution are identified and the solution obtained. Four slightly different block algorithms are tested and evaluated. The comparison include the number of factorizations, time, and the numerical accuracy of the solution. The solution of rank deficient problems is also discussed.

In Section 6 the bounded least squares problem is tackled with a special interior-point method first proposed by Mehrotra. We use the logarithmic barrier function to introduce the interior-point method used. The method by Mehrotra is a predictor-corrector method with a clever way of estimating the barrier parameter. Different schemes to use multiple corrections are also discussed and numerical results given. The rank deficient case is solved by adding a regularization term. We try an adaptive scheme for decreasing the regularization parameter as we converge toward the solution.

In Section 7 the two different classes of methods from Section 5 and Section 6 are compared.

1.2 Constrained optimization

In most applications of mathematical models there is a natural underlying constraint, due to the finite resources in nature. These constraints do not always interfere with the solution but when they do one must use a method that ensures that the constraints are not violated at the solution. In constrained optimization the Karush-Kuhn-Tucker conditions are used to characterize the solution. We will here present some useful results in constrained optimization of convex functions.

1.2.1 Some notations and background

In this section we will describe some of the notations commonly used in optimization contexts. Our goal is to find the optimal value of a given *objective*

function, $f(x)$ in a given domain, $\Omega \in I\!R^n$. By the optimal value we mean the maximal or minimal value in the domain. A maximization problem can easily be rewritten as a minimization problem by substituting $f(x)$ by $-f(x)$. If the domain is equal to the whole space we have an *unconstrained* optimization problem otherwise a *constrained* problem. We will assume that the domain Ω is a true subset of $I\!R^n$, nonempty, and closed. The domain Ω can be either bounded or unbounded.

A point in Ω is called a *feasible* point and *strictly feasible* if it belongs to the interior of Ω. The domain Ω is usually defined by some set of constraints. These can be equality constraints, $c_i(x) = 0$ or inequality constraints, $c_i(x) \geq 0$. At a given point $x \in \Omega$ the constraints can be divided into two different sets, the constraints that hold with equality, called *active* and the constraints which holds with inequality, called *free*. If an active constraint is redundant, i.e., we get the same solution with the constraint removed, we have a *degenerate* solution.

A *feasible direction* v at a point $x \in \Omega$ is a direction such that $x + \alpha v$ is feasible for a infinite sequence $\alpha_i \to 0$, $\alpha_i > 0$. Sometimes the zero vector is included and this subspace of vectors $x + \alpha v$ is referred to as the tangent cone at x. This means that $c_i(v) \geq 0$, $\forall i \in \mathcal{B}$ and with equality if we have equality constraints, for all $i \in \mathcal{B}(x)$.

Linear constraints. If the constraints are linear they can be described by a matrix $C \in I\!R^{m \times n}$ and a vector $d \in I\!R^m$ such that

$$\Omega \equiv \{x \in I\!R^n \mid Cx - d \geq 0\} \tag{1.2}$$

(m is the number of constraints and n the number of variables). Each row $c_i^T x - d_i \geq 0$ describe a half-space in $I\!R^n$ with c_i^T as the normal to the hyper-plane $c_i^T x - d_i = 0$. This hyper-plane can be viewed as a facet of the polytope defined by $Cx - d \geq 0$. If (1.2) is nonempty then $Cx - d \geq 0$ is called consistent. This set has an important property namely convexity. Convexity can be defined as follows (there are more general ways to define convexity, but for our purposes this definition is enough).

DEFINITION 1.1 *A set $\Omega \neq \emptyset$ is called convex if*

$$z = \lambda x + (1 - \lambda)y \in \Omega,$$

for all for all $x, y \in \Omega$ and $\lambda \in [0\ 1]$.

DEFINITION 1.2 *A function $f(x)$ is called convex in Ω if*

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda) f(y), \tag{1.3}$$

for all for all $x, y \in \Omega$ and $\lambda \in [0\ 1]$. The function $f(x)$ is strictly convex if (1.3) holds with strict inequality.

It is easy to verify that a set Ω defined by linear inequalities is convex. An optimization problem with a convex set Ω and a convex objective function $f(x)$, is called a *convex programming* problem. The optimal point to an optimization problem can be either a *local* or *global* optimum.

DEFINITION 1.3 (LOCAL (CONSTRAINED) MINIMIZER) *The point x^* is a local minimizer of the problem* $\min f(x)$ *subject to (1.2), if there exists a compact set $\mathcal{S}$ such that*

$$x^* \in \text{int}(\mathcal{S}) \cap \Omega \quad \textit{and} \quad f(x^*) = \min\{f(x) \mid x \in \mathcal{S} \cap \Omega\}.$$

The minimizer is strict if there is no other point y in Ω such that $f(x^*) = f(y)$.

DEFINITION 1.4 (GLOBAL (CONSTRAINED) MINIMIZER) *The point x^* is a global minimizer of the problem* $\min f(x)$ *subject to (1.2), if*

$$x^* \in \Omega \quad \textit{and} \quad f(x^*) = \min\{f(x) \mid x \in \Omega\}.$$

The global minimizer is strict if $f(x^*) < f(y)$ for all $y \neq x^* \in \Omega$.

The local and global optimal points to a convex programming problem has the following property.

THEOREM 1.1 *Consider a convex programming problem* $\min_{x\in\Omega} f(x)$. *If x^* is a local optimal point then x^* is also a global optimum. Further, if $f(x)$ is strictly convex then x^* is the unique optimum. The set of optimal solutions is convex.*

Proof: Suppose there are two different local optima x and y. By the convexity we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y).$$

But x and y are local optima so equality must hold and $f(x) = f(y)$ or we have a contradiction.

If f is strictly convex we have a strict inequality and thus get a contradiction unless $x = y$ is then the global unique optimum.

Denote the solution set as $\mathcal{S} = \{y \mid f(y) = \min\}$. Take $x_1, x_2 \in \mathcal{S}$, and let $\hat{x} = \lambda x_1 + (1-\lambda)x_2$, $\forall \lambda \in [0,\ 1]$. By the convexity;

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2) = \min f(x).$$

Therefore $\hat{x} \in \mathcal{S}$, x and y arbitrary in $\mathcal{S}$. It follows that $\mathcal{S}$ is convex. □

To obtain a unique global optimal point when $f(x)$ is not strictly convex, one can impose the following condition on the solution.

COROLLARY 1.2 *The problem*

$$\min_{x\in\mathcal{S}} g(x) = ||x||_2, \quad \mathcal{S} = \{x \mid f(x) = \min\},$$

from the convex programming problem above, has a unique minimum.

Proof: $g(x)$ is a strictly convex function, bounded below. $\mathcal{S}$ is convex and the minimum point is unique. □

An important aspect of convex programs is that the the first order necessary conditions are sufficient to determine a global optimum. This is important for the interior and active set methods we derive. To show that our least squares problem is convex we use the equivalent formulation as a quadratic programming problem.

LEMMA 1.3 *The function $f(x) = \frac{1}{2}||Ax - b||_2^2$ is a convex function in $\mathbb{R}^n$ and strictly convex if and only if A has full column rank.*

Proof: We want to show that the convexity definition, $\lambda f(x) + (1-\lambda)f(y) - f(\lambda x + (1-\lambda)y) \geq 0$ holds for all $\lambda \in [0\ 1]$ and $x \neq y$. By the definition of $||\cdot||_2$, $f(x) = \frac{1}{2}x^T A^T A x^T - x^T A^T b$

$$\begin{aligned}
&\lambda(\tfrac{1}{2}x^T A^T A x - x^T A^T b) + (1-\lambda)(\tfrac{1}{2}y^T A^T A y - y^T A^T b) \\
&\qquad - (\tfrac{1}{2}\lambda^2 x^T A^T A x - \lambda x^T A^T b + \tfrac{1}{2}(1-\lambda)^2 y^T A^T A y \\
&\qquad - (1-\lambda)y^T A^T b) + \lambda(1-\lambda)x^T A^T A y \qquad (1.4)\\
&= \tfrac{1}{2}\lambda(1-\lambda)(x^T A^T A x + y^T A^T A y - 2x^T A^T A y) \\
&= \tfrac{1}{2}\lambda(1-\lambda)||A(x-y)||_2^2 \geq 0.
\end{aligned}$$

If A has full column rank, $x \neq y$ and $\lambda > 0$ the last term will be strictly greater than 0. Suppose $f(x)$ strictly convex and let $y = 0$ and $x \in \mathbb{R}^n$ in (1.4). Then we get,

$$\tfrac{1}{2}\lambda(1-\lambda)||Ax||_2^2 > 0,\ \forall x \neq 0 \in \mathbb{R}^n \Leftrightarrow A \text{ has full column rank.}$$

□

For further information on convex functions and sets, we refer to Mangasarian [46] or the classical book by Rockafellar [63].

1.2.2 Optimality conditions

Assume that the objective function is twice continuously differentiable at x^*. Then the necessary conditions for the point x^* to be a local minimizer is as follows:

THEOREM 1.4 (UNCONSTRAINED LOCAL MINIMUM) *The point x^* is a local minimum to $f(x)$, f two times differentiable, if*

$$\nabla f(x^*) = 0, \text{ and } H(x^*) \text{ positive semidefinite,}$$

where ∇f is the gradient and $H(x)$ is the Hessian to f ($H_{i,j} = \frac{\partial^2 f}{\partial x_i \partial x_j}$).

A point x^* satisfying the gradient condition $\nabla f(x^*) = 0$ is called a *stationary point.*

The optimality conditions for a constrained problem must also take into account the constraints. Assume that the function $f(x)$ is continuously differentiable, and the domain Ω is determined by inequality constraints. Then if ∇f is in the convex cone spanned by the normals to the active constraints, there cannot be any descent direction. This is essentially Farkas' Lemma. It is convenient to introduce the *Lagrangian function.*

$$L(x, \lambda) = f(x) - \sum_{i=1}^{n} c_i(x)\lambda_i,$$

when stating the necessary optimality conditions. An optimal point to $f(x)$ is a saddle point to the Lagrangian function and the first order condition (often called Karush-Kuhn-Tucker (KKT) conditions) is stated in Theorem 1.5.

THEOREM 1.5 (OPTIMALITY CONDITIONS FIRST ORDER (KKT)) *If x^* is a local minimizer to $f(x)$ subject to $Cx - d \geq 0$, then there exists multipliers λ^* such that x^*, λ^* satisfies the following system:*

$$\begin{aligned} \nabla_x L(x, \lambda) &= 0, \\ Cx - d &\geq 0, \\ \lambda_i(c_i^T x - d_i) &= 0, \ \forall i, \\ \lambda_i &\geq 0, \ \forall i. \end{aligned} \tag{1.5}$$

c_i^T is the ith row of C.

The multipliers λ_i^* above are often refereed to as *Lagrangian multipliers.* The condition $\lambda_i(c_i^T x - d_i) = 0$ is called *complementarity* condition. This means that if $c_i^T - d > 0$ the i constraint is *inactive* and $\lambda_i^* = 0$. If the constraint and the multiplier is zero at the same time, the solution is *degenerate.*

In unconstrained optimization a local optimum is strict if and only if the Hessian is positive definite. However in constrained optimization it is only the curvature in the feasible directions that is relevant. Therefore the *restricted Hessian* is positive definite if $s^T H s > 0$ in all feasible directions s. This forms the sufficient optimality condition in Theorem 1.6. These conditions can be reformulated to cover the general case with equality constraints. The first order conditions for inequality constraints are formulated as follows.

In case A has full column rank these conditions are always true, and we have a unique optimum (as shown above). But even if A does not have full column rank the optimum may be unique.

THEOREM 1.6 (SUFFICIENT OPTIMALITY CONDITIONS OF SECOND ORDER) *If the first order optimality conditions at x^* hold and*

$$s^T H(x^*)s > 0, \qquad \forall s \in \{v \mid v \text{ feasible direction at } x^*\}$$

then x^* is a strict local minimum.

Proofs of these theorems can be found in basic textbooks on constrained optimization, Fletcher [17, Chap. 9], Gill, Murray and Wright [31, Chap. 7].

1.2.3 Quadratic programming problems

The problem we are looking at is a special case of a wider class of problems, the general quadratic programming (QP) problem,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & q(x) = \frac{1}{2}x^T G x + g^T x, \\ \text{subject to} \quad & Cx \leq d, \\ & Ex = f, \end{aligned} \tag{1.6}$$

where G is a symmetric matrix, definite or indefinite, and the relative interior of the feasible domain is non-empty. If G is positive definite or positive semidefinite, a bounded global optimal solution exists. If G is positive definite the optimum is also unique. This follows from the convexity of $q(x)$ and the constraints. If we let $G = A^T A$ and $g = -A^T b$ we will obtain the the constrained Least Squares Problem in the quadratic formulation. G is then positive semidefinite and the problem has a bounded global optimum. The quadratic programming problem with an indefinite matrix G is much harder to solve. Even if only one eigenvalue has different sign the problem is in NP, see [67]

The problem (1.6) has been studied thoroughly in the literature and various approaches developed. Two of the mostly common approaches are active set methods and interior-point methods. The active set methods for general QP problems with general bounds have been treated in [17, 30, 29, 31, 56]. Active set methods for quadratic programming problems with simple bounds are developed in Section 5. General interior-point methods for QP can be found in [8, 53, 59, 68, 69]. Other interesting methods that are closely related to block active set methods are the projected gradient methods [7] and the reflective Newton method [10]. These methods apply to optimization problems with a general quadratic function and bounds on some variables.

1.2.4 Optimality conditions for quadratic programming problems

The optimum point of (1.6) can be characterized by the first order conditions, the Karush-Kuhn-Tucker (KKT) conditions. The second order conditions include the Hessian G of $q(x)$ which is by our assumptions positive definite or positive semidefinite. In either case, as shown, the stationary point that satisfies KKT condition is also a global optimal point, but not necessary unique if G is positive semidefinite.

1.3 General quadratic programming software

There is a great commercial interest in codes for solving linear and quadratic programming problems. This is reflected in the the amount of commercial codes

that are available on the net. Several of the codes are distributed by software vendors closely connected to universities. The packages listed in Table 1.1, are solvers for constrained linear or quadratic problems. One could use a more general solver for the problem studied here, a nonlinear optimization code, but this will lead to a great loss of efficiency due to the fact that one then neglects important knowledge of the aspects of the quadratic problem.

In this thesis we look at a quadratic programming problem with a special structure, a least squares problem. If the algorithms are implemented properly one can avoid forming the normal equations $A^TAx = A^Tb$ and therefore gain accuracy in the solution of ill-posed problems.

In Table 1.1 and Table 1.2 below is an overview of existing quadratic programming software. This information can also be found at *NEOS Guide: Optimization software* at URL:
`http://www.mcs.anl.gov/home/otc/Guide/SoftwareGuide/`. This www site is based on the book by Coleman and Li [57], but moved to a web-site in order to always be up-to-date.

Table 1.1: Solvers for QP problems, description and authors.

Package	Description	Author
BQPD	quadratic programming.	R. Fletcher
CPLEX	linear, quadratic, and network linear programming.	
LINDO	linear, mixed-integer and quadratic programming.	
LOQO	linear, quadratic programming	R. J. Vanderbei
LSSOL	least squares problems.	
OSL	linear, quadratic and mixed-integer programming.	
PORT 3	minimization, least squares, etc.	
SQOPT	large-scale linear and convex quadr progr.	Gill, Murray, Saunders
SNOPT	large-scale linear, quadr. and nonlinear progr. problems (including nonconvex quadratic progr.)	Gill, Murray Saunders
QL	convex quadratic programming.	K. Schittkowski
QPOPT	linear and quadratic problems.	W. Murray, P. E. Gill
OPTIM	linear and quadratic programming and contrained least squares problems	MATLAB

Table 1.2: Solvers for QP problems, method and distribution.

Package	Method	Distribution
BQPD	active set	Univ. of Dundee
CPLEX	prim, dual simplex, interior-point	CPLEX Optimization, Inc.
LINDO	simplex, active set	LINDO Systems, Inc.
LOQO	interior-point	Princeton University
LSSOL	active set	Stanford Business Software, Inc.
OSL	prim, dual simplex, interior-point	OSL Development, IBM Corp.
PORT 3	trust region, interior-point active set	netlib
SQOPT		Stanford Business Software, Inc.
SNOPT		Stanford Business Software, Inc.
QL	dual method of Goldfarb, Idnani	Univ. of Bayreuth
QPOPT		Univ of California, San Diego
OPTIM	simplex, active set	MathWorks

2 The linear least squares problem

In many applications there is a mathematical model describing the physical aspects involved. Measurements of some kind are made to determine the parameters of the model. Errors are introduced by the measurement method. By taking more samples than needed the influence of errors on the solution can be reduced. The system of equations that arises is overdetermined, i.e. it has more equations than variables. The method for solving the overdetermined system should give a "better" answer in some sense, if more measurements exist. The least squares solution minimizes the residual in the 2-norm, and is the best linear unbiased estimate of the solution to the overdetermined system, see [5].

In this section methods for solving the linear least squares problem,

$$\min_x ||Ax - b||_2, \quad A \in I\!R^{m\times n},\ b \in I\!R^m, \tag{2.1}$$

will be introduced. Only direct methods based on the QR factorization of A or the Cholesky factorization of the matrix of normal equations will be discussed.

The problem of computing the solution with the minimal norm to an underdetermined system will also be discussed. The matrix A in this section will be assumed to be dense. The sparse case will be discussed in the next section. For a more complete treatment of the subject Björck [5] and Lawson and Hanson [38].

2.1 History of the sum of residuals

The principle of linear least squares for solving a overdetermined system of equations has been known for a long time. The method of least squares was first published by Legendre 1805 in a paper entitled "Nouvelles méthodes pour la détermination des orbites des cométes". Gauss proved the statistical properties of the solution in 1809, and remarked at the same time that he knew of the method already 1795 (at the age of 18). Most historians think that Gauss has right to his claim because of his predictions concerning the comet Ceres in 1801.

Gauss wrote two memoirs in 1821 and 1823 which treats the least squares problem and its statistical properties. The memoirs have been translated into English by Stewart [19]. Gauss proves here a theorem about the optimality of the least squares estimate without any assumptions about the distribution of the random variables. This theorem was later rediscovered by Markoff in 1912.

The method of normal equations was the standard numerical method used to compute the least squares estimate until 1965 when Golub [32] proposed solving the least squares problem by orthogonal Householder transformations [36].

2.2 Solving unconstrained least squares problems

Consider the linear system $Ax = b$, $A \in I\!R^{m\times n}$. This is an overdetermined problem when $m > n$. If $b \notin \mathcal{R}(A)$ a solution to this system does not exist, $\mathcal{R}(A)$ denotes the range of A. Later the null space of A will be denoted $\mathcal{N}(A)$.

However, a solution that minimizes the residual $r = b - Ax$ in the 2-norm can always be found. The following theorem will characterize the solutions by a simple orthogonality relation.

THEOREM 2.1 *Let the set of all solutions to the least squares problem (2.1) be equal to $\mathcal{S}$, then*

$$x \in \mathcal{S} \quad \Leftrightarrow \quad A^T(b - Ax) = 0.$$

The set $\mathcal{S}$ is not empty.

Proof: The usual way to prove this is an algebraic proof by contradiction. Here a proof based on the optimality condition in Theorem 1.4 is given. If we let

$$f(x) = \frac{1}{2}||Ax - b||_2^2 = \frac{1}{2}x^T A^T Ax - x^T A^T b + \frac{1}{2}b^T b,$$

then $\nabla f(x) = A^T(Ax - b)$ and $H(x) = A^T A$. From Theorem 1.4 it follows that x is a local minimizer if

$$\nabla f(x) = 0 \;\Leftrightarrow\; A^T(b - Ax) = 0 \Leftrightarrow\; A^T Ax = A^T b.$$

The last system of equations is called the *normal equations* and is always consistent since $A^T b \in \mathcal{R}(A^T) = \mathcal{R}(A^T A)$. The set of stationary points ($\mathcal{S}$) are therefore not empty. The second condition in Theorem 1.4, $x^T Hx = ||Ax||_2^2 \geq 0$, is trivially satisfied. The points in $\mathcal{S}$ are local and also global minimizers to $f(x)$ due to the convexity of $f(x)$. □

The geometric interpretation of this characterization of a solution $x \in \mathcal{S}$ is that the residual $r = b - Ax^*$, is orthogonal to all vectors in the subspace $\mathcal{R}(A)$.

2.2.1 Sensitivity of the least squares solution

Due to the limitations of floating point arithmetic the data A and b can not be represented exactly and the computed solution will have an error that depends on the floating point system used. In the floating point system *IEEE double precision* used in the numerical computations here, a real number can be represented with a relative error no larger than $\mathbf{u} = \frac{1}{2}\beta^{t\perp 1}$, $\beta = 2$ and $t = 53$. This quantity is called *unit roundoff*, [35, p. 42]

By the *forward error* in $y^* = f(x) + \delta$ we mean the absolute error δ in y^* that occurred during the computation of $f(x)$ in finite arithmetic. This can often be bounded by some function times the *unit roundoff*, $\mathbf{u}$. A *backward error* is the smallest error in x that in exact arithmetic produces the same result $y^* = f(x + \delta x)$. If we can not store the input data exact, we have to be satisfied if the solution to our problem is the exact solution to a nearby problem (with a deviation bounded in some norm by a constant times $\mathbf{u}$). For a *normwise backwards stable* method this property is fulfilled for all problems in a specific class. The output of a backward stable method is the exact solution to a nearby problem in the same problem class with input data perturbed by a factor $\mathcal{O}(u)$.

A *forward error stable* algorithm is usually defined by the requirement that the relative error in the solution is bounded by κu, where κ is a condition number of the problem.

The condition number of a rectangular matrix can be defined as follows.

DEFINITION 2.1 *The condition number of* $A \in \mathbb{R}^{m\times n}$ $(A \neq 0)$ *is defined as*

$$\kappa(A) = \sigma_{max}/\sigma_{min},$$

where σ_{max} *and* σ_{min} *are the largest and smallest positive singular value to* A.

With this definition the sensitivity of the solution to (2.1) in regard to errors in A and b can be formulated as follows.

THEOREM 2.2 *Assume that* $\text{rank}(A+\delta A) = \text{rank}(A)$, *and let*

$$\frac{||\delta A||_2}{||A||_2} \leq \epsilon_A, \quad \frac{||\delta b||_2}{||b||_2} \leq \epsilon_b.$$

Then if $\eta = \kappa\epsilon_A < 1$ *the perturbation* δx *and* δr *in the least squares solution* x *and the residual* $r = b - Ax$ *satisfy*

$$||\delta x||_2 \leq \frac{\kappa}{1-\eta}\left(\epsilon_A||x||_2 + \epsilon_b\frac{||b||_2}{||A||_2} + \epsilon_A\kappa\frac{||r||_2}{||A||_2}\right) + \epsilon_A\kappa||x||_2, \tag{2.2}$$

and

$$||\delta r||_2 \leq \epsilon_A||x||_2||A||_2 + \epsilon_b||b|| + \epsilon_A\kappa||r||_2.$$

The last term in (2.2) vanishes if A *has full rank.*

Proof: See Björck [5] p. 31. □

In the error bound for δx we have a term proportional to $\kappa^2||r||_2$. This term shows that the error in the solution do not only depend on the matrix A as in the linear equation case, but also on the right hand side b. When A has the full column rank and $\epsilon_b = 0$, the condition number for the least squares system can be written as

$$\kappa_{LS}(A, b) = \kappa(A)\left(1 + \kappa(A)\frac{||r||_2}{||A||_2||x||_2}\right).$$

The error in the solution can therefore become very large if the residual is large.

2.2.2 Normal equations

The most common way to solve the least squares problem is to form the normal equations $A^TAx = A^Tb$ and solve this system for x. If A has full rank the normal equations can be solved by Gaussian elimination applied to A^TA without row and column interchanges in a numerically stable way. By using the symmetric and positive definite nature of the matrix A^TA the work in the factorization can

be reduced approximately with a factor two using a Cholesky factorization of A^TA. The procedure for solving the least squares problem then becomes,

$$\begin{aligned} C = A^TA,\ d = A^Tb, \\ R^TR = C,\ R \text{ triangular}, \\ R^Ty = d, \\ Rx = y. \end{aligned}$$

This solution process requires $n^2(m + n/3)$ flops. By forming the cross-product A^TA, the matrix A is compressed from a $m \times n$ to a symmetric $n \times n$ system. However, in the process of forming this cross-product backward stability for the least squares problem is lost. It is easy to show[1] that $\kappa(A^TA) = \kappa^2(A)$; therefore some of the information is lost when forming A^TA. The Cholesky method may fail already when $\kappa(A)$ is close to $1/\sqrt{\mathbf{u}}$.

Solving symmetric, positive definite linear system with the Cholesky factorization is a backwards stable method. The computed solution x^* satisfies,

$$(C + E)x^* = A^Tb, \quad ||E||_2 \leq 2.5n^{3/2}\mathbf{u}||A||_2^2,$$

with the error bound,

$$||x^* - x||_2 \leq 2.2n^{3/2}\mathbf{u}\kappa^2(A)||x||_2. \tag{2.3}$$

However, as pointed out before, this does not give a backward stable method for the least squares problem. By carefully studying the proof of the error bound above Björck found that one factor $\kappa(A)$ in the bound above can be replaced with the "column scaled" condition number $\kappa'(A) = \min_D \kappa(AD)$, where D is a diagonal matrix with positive elements. Observe that this scaling does not have to be carried out on the matrix A before doing the Cholesky factorization. This is important in the interior-point methods where we get large scaling factors when the method converges to the solution.

The solution obtained from the normal equations can be improved by making fixed precision iterative refinement,

$$\begin{aligned} r = b - Ax, \\ R^T(R\delta x) = A^Tr, \\ x \leftarrow x + \delta x. \end{aligned}$$

When the iterative refinement is carried out the error bound of (2.3) is initially reduced by a factor proportional to

$$\delta \sim \kappa^2(A)\mathbf{u},$$

for each iteration. The refinement will converge until the errors from computing the residual r is the dominating factor. If the residual is computed in higher

[1] By using the singular value decomposition, $A = U\Sigma V^T$.

precision this error is not dominant and the refinement will often converge to a solution accurate to full machine precision.

The LAPACK routine `_POSV(X)` solves a dense symmetric positive definite system with Cholesky factorization, see [2].

2.2.3 Methods based on QR factorization

In the last section we remarked that in forming the matrix product A^TA and A^Tb, information was lost and so was the backward stability. This can be avoided by using orthogonal transformations when solving the least squares problem. One can show with a constructive proof that A can be transformed by an orthogonal transformation from the left to a triangular matrix R. We write the QR factorization

$$A = (Q_1 \; Q_2) \begin{pmatrix} R \\ 0 \end{pmatrix}, \tag{2.4}$$

where $(Q_1 \; Q_2)$ is orthogonal, $Q_1 \in I\!R^{m\times n}$, $Q_2 \in I\!R^{m\times(m\perp n)}$ and $R \in I\!R^{n\times n}$. The following important relationships hold, $\mathcal{R}(Q_1) = \mathcal{R}(A)$ and $\mathcal{R}(Q_2) = \mathcal{N}(A^T)$. There are several ways to compute the QR factorization for dense matrices. It is usually carried out by Householder transformations

$$P = I - \frac{2}{u^Tu}uu^T.$$

The cost for computing the QR factorization by Householder transformations is $2n^2(m - n/3)$ if Q is not explicitly computed but stored as the Householder vectors. This is twice the work for the Cholesky factorization if $m \gg n$. The R obtained in exact arithmetic by QR is identical to R from Cholesky except from possible sign changes of the rows. This follows from $A^TA = R^TQ^TQR = R^TR$ and the essential uniqueness of the Cholesky factorization.

By using the QR factorization together with the invariance of the 2-norm[2] the following equivalent problem can be obtained,

$$\min_x ||Q^T(Ax - b)||_2 \Leftrightarrow \min_x ||Rx - d_1||_2 + ||d_2||_2,$$

where $Q^Tb = (d_1^T \; d_2^T)^T$. The solution is obtained by solving the triangular system, $Rx = d_1$ and the residual norm is given by $||d_2||_2$.

If $\overline{x}$ is the computed solution to the least squares problem solved by Householder QR it is possible to show that $\overline{x}$ is the solution to the nearby problem

$$\min_x ||(A + \delta A)x + (b + \delta b)||_2,$$

where the perturbations satisfy

$$||\delta A||_2 \leq c\mathbf{u}n^{1/2}||A||_2, \qquad ||\delta b||_2 \leq c\mathbf{u}||b||_2,$$

[2] $||Qx||_2^2 = x^TQ^TQx = ||x||_2^2$, Q orthogonal.

c is a low order polynomial in m and n, see Lawson and Hanson [38]. The solution is therefore normwise backwards stable.

In some large-scale applications the orthogonal matrix Q can not be stored efficiently. If the right-hand side is not known at factorization time then the product $Q^T b$ can not be computed. However, if the original A is still available then the *semi normal equations* (SNE),

$$R^T R x = A^T b,$$

R from QR, can be used. Even though $A^T A$ is not formed, this equation does not have any better stability properties than the normal equations. However, if the SNE is used together with one step of iterative refinement we obtain the *corrected seminormal equations* (CSNE).

$$\begin{aligned} R &= Q^T A, \\ R^T(Rx) &= A^T b, \\ r &= b - Ax, \\ R^T(R\delta x) &= A^T r, \\ x_c &= x + \delta x. \end{aligned}$$

An error analysis of this combination was done by Björck [3] and the error in x_c computed from CSNE satisfies,

$$||x - x_c|| \leq c_1 \mathbf{u}\kappa^2(A)\,(c_2 \mathbf{u}\kappa(A))\,||x||_2.$$

If $c\mathbf{u}\kappa^2(A) < 1$ this is no worse that the backward error of the QR factorization. The error bound can be sharpened by replacing $\kappa^2(A)$ with $\kappa(A)\kappa'(A)$ and the last κ by κ', $\kappa' = \min_D \kappa(AD)$, D is a diagonal matrix with $d_{ii} > 0$. The rate of convergence if more steps of iterative refinement is carried out is proportional to

$$\delta \sim \mathbf{u}\kappa'(A).$$

This is a factor $\kappa(A)$ better than the convergence rate of iterative refinement with R computed by the Cholesky factorization.

2.3 The underdetermined system

A problem closely related to the least squares problem is finding the minimum norm solution of the underdetermined system $A^T y = c$, $A \in \mathbb{R}^{m\times n}(m > n)$. If $c \in \mathcal{R}(A^T)$ then the system has infinite number of solutions. To obtain a unique solution, we consider to the solution of smallest 2-norm

$$\min_y ||y||_2, \quad A^T y = c. \tag{2.5}$$

A perturbation analysis can be obtained in the same manner as for the least squares problem.

THEOREM 2.3 *Suppose* $\text{rank}(A) = n \leq m$ *and* $c \neq 0$ *and let*

$$\epsilon_A = \frac{||\delta A||_2}{||A||_2}, \quad \epsilon_b = \frac{||\delta b||_2}{||b||_2}.$$

If $\epsilon = \max\{\epsilon_A, \epsilon_b\} < \sigma_{min}(A)$ *then the error* δy *in the solution will satisfy*

$$||\delta y||_2 \leq \kappa(A)(\epsilon_A \min\{2, m-n+1\} + \epsilon_b)||y|| + \mathcal{O}(\epsilon^2).$$

Proof: See Golub and van Loan [33], p. 273. □

Here there is no κ^2 term as in the least squares problem.

If the problem (2.5) is consistent the solution will satisfy the *normal equations of the second kind*,

$$A^T A z = c, \quad y = Az. \tag{2.6}$$

If A has full rank (2.6) can be solved by a Cholesky factorization. If a QR factorization (2.4) is available we can compute the solution from

$$R^T w = c, \quad y = Q_1 w.$$

This method is backward stable [35]. Finally if the orthogonal matrix can not be stored a similar version for the semi-normal equations can be used,

$$R^T R z = c, \quad y = Az.$$

An error bound for SNE that depends on $\kappa(A)$ and not $\kappa^2(A)$ as in the least squares has been proved by Paige [61]. The bound on the solution is,

$$||y - y^*||_2 \leq c\mathbf{u}\kappa(A)||y||_2,$$

where $c = c(m,n)$ is a polynomial of low degree. However, this method is not backward stable, and the residuals can be of order $\kappa(A)\mathbf{u}$.

By iterative refinement as in the least squares case the accuracy of a computed solution can be improved. The iterative refinement with the normal equations of the second kind should be done in the following way, Björck and Paige [6, ALGORITHM 7.3],

$$\begin{aligned} s &= c - A^T x, \\ R^T(R\delta y) &= -s, \\ x &\leftarrow x - A\delta y. \end{aligned} \tag{2.7}$$

The following observation will be used when constructing the test problems in section 4.

OBSERVATION 2.4 *The minimum norm solution to the underdetermined system* $A^T y = c$ *is unchanged if we instead minimize*

$$\min_y ||y - Ax||_2, \quad A^T y = c, \quad x \in \mathbb{R}^n \text{ arbitrary.} \tag{2.8}$$

Proof: Let y^+ be the minimum norm solution to $A^T y = b$. The solution y to (2.8) can be written as $y = y^+ + u$, $A^T y = c + A^T u \Rightarrow A^T u = 0 \Leftrightarrow u \in \mathcal{N}(A^T) = \mathcal{R}(A)^\perp$.

$$\begin{aligned} ||y - Ax||_2^2 &= ||y^+ - Ax||_2^2 + ||u||_2^2 + 2u^T y^+ - 2x^T A^T u = \\ &= ||y^+ - Ax||_2^2 + ||u||_2^2. \end{aligned} \tag{2.9}$$

Here $u^T y^+ = 0$ because $u \in \mathcal{R}(A)^\perp$ and $y^+ \in \mathcal{R}(A)$ from the normal equations of the second kind. Hence (2.9) is minimized when $u = 0$. □

3 Sparse matrix computation

An early characterization of a sparse matrix was given by Wilkinson; a sparse matrix is a matrix with enough zeros so its advantageous to use this sparsity. In sparse matrix computation the *sparsity pattern*, the pattern of non zero elements, is analyzed and an ordering of the columns/rows is made so the number of new elements appearing during the calculation and the storage requirements are minimized.

The simplest class of sparse matrices is the class of banded matrices. To solve a banded $n \times n$ linear system it requires about $2npq$ flops[3] where p and q is the number of diagonals over respectively under the main diagonal and n is the size of the system. The saving in computational effort compared to a dense solver, which requires $2n^3/3$ flops, can be enormous when n is large.

In sparse techniques for solving linear systems there is a trade-off between the amount of fill-in, i.e. the non zero elements introduced during the computation, and the stability issue. For example, when solving a linear system of equations by LU factorization the fill-in in L and U is strongly dependent on the column and row ordering. However, from a numerical point of view the optimal ordering can be a bad choice. In our application to the least squares problem the row order does not influence the fill-in in the final factorization, but we must choose a good column ordering to minimize the fill-in. The choice of column permutation of A does not influence the stability of the QR factorization. The same is true for the Cholesky factorization.

An introduction to sparse techniques is given by Duff et al. [14] and by George and Liu [23]. The later treats computational methods for solving sparse positive definite systems.

3.1 Sparse storage schemes

There are several ways to store a sparse matrix. The most efficient way to store the matrix is dependent on the structure of the matrix and in what context the matrix is used. The storage scheme plays an important role for the speed of manipulating the matrix. For example, the most straightforward storage scheme for a matrix is to use a *coordinate scheme*. For each nonzero element in A we store the triple (a_{ij}, i, j). This requires nnz reals and $2 \cdot nnz$ integers, where nnz is the number of nonzero elements in A. The main drawback, except for the expensive memory requirement, is the difficulty of retrieving a specific element. If the elements are not sorted in any order, we have to search through all elements to decide if an element is zero. The only good thing about this storage scheme is that it's easy to move the sparse matrix between different numerical packages. Then a conversion can be made to a storage scheme suitable for the application.

Another storage scheme currently used in MATLAB V5, see Gilbert, Moler and Schreiber [26], is the *compressed sparse column* scheme. In this storage

[3] Provided no pivoting are done.

scheme each column in A is compressed into a dense array. These dense arrays are stored in a one dimensional vector, VAL. To each element, the row index is stored in a integer array IRN. The third array of n+1 integers JPTR. JPTR holds pointers to the first element in each column. It is often convenient to let the $n+1$ element in JPTR be equal to nnz. The total storage needed in this scheme is nnz reals and $nnz+n+1$ integers. This scheme could as easily be used on the rows instead of the columns.

The first three columns of matrix in Figure 3.1 is represented in the sparse compressed column scheme in Table 3.1.

Table 3.1: The compressed column scheme for storing a sparse matrix.

VAL	a_{11}	a_{21}	a_{61}	a_{12}	a_{22}	a_{32}	a_{62}	a_{23}	a_{33}	a_{43}	a_{53}
IRN	1	2	6	1	2	3	6	2	3	4	5
JPTR	1	4	8	12							

A technique to speed up certain computations is used in MATLAB . When a column is used in a computation the compress column is expanded into a full vector, or the sparse accumulator. Now each element can be accessed in constant time and if the routines are built up by SAXPY[4] operations they can be very efficient. When the column is not needed anymore it is compressed and stored in the sparse format. The fact that the matrix is stored in column-wise order is important when implementing sparse algorithms in MATLAB. The access time can be up to one magnitude slower if the matrix is accessed by rows instead of columns.

3.2 Graph representation

An important tool to analyze the sparsity pattern of a sparse matrix is graph theory. A general sparse matrix can be represented by a directed graph $G(X, E)$ consisting of a set of numbered nodes, X and the set of edges between the nodes, E. If the entry $a_{ij} \neq 0$ then the edge (i, j) from node n_i to n_j belongs in the set E. An edge between nodes usually is marked with a arrow and is therefore called a directed graph. If A is symmetrical each edge will be double, one in each direction. Therefore we can discard the direction of the edges and then look at an undirected graph. Figure 3.1 gives an example of a symmetric matrix and its graph representation.

By analyzing the graph connected with a matrix we can order the calculation to minimize unnecessary fill-in during the calculation.

3.3 Sparse orderings

The order of the rows/columns is very important in sparse computations. When solving a dense linear system the row and column ordering is usually chosen so

[4] A SAXPY operation is $ax + y$, a scalar x, y vectors

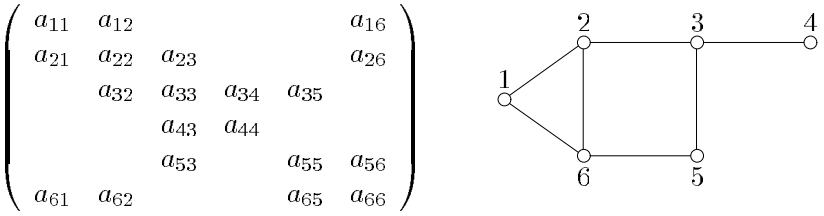

Figure 3.1: Graph representation of a sparse matrix. The numbers corresponds to the column of the matrix.

the numerical solution process is stable. In sparse computations the ordering is often chosen to minimize fill-in. An example of this is the following matrix,

$$\begin{pmatrix} \times & \times & \times & \times \\ \times & \times & 0 & 0 \\ \times & 0 & \times & 0 \\ \times & 0 & 0 & \times \end{pmatrix} \qquad \begin{pmatrix} \times & 0 & 0 & \times \\ 0 & \times & 0 & \times \\ 0 & 0 & \times & \times \\ \times & \times & \times & \times \end{pmatrix}$$

Original order Reversed order

If the system of equations is solved with the original ordering, all zeros will be destroyed in the intermediate process. On the contrary, if the columns and rows are ordered in reverse order, there will be no fill-in when solving the system. However, the numerical process may not be stable and therefore some threshold parameter is often introduced to relax the pivot strategy from dense matrix computation. When solving symmetrical positive definite systems we can always use symmetrical pivoting without affecting the numerical numerical stability.

The problem of computing the best ordering in the sense of least fill-in is an NP-complete problem. Therefore only an approximation of the minimum fill-in can be computed for large problem. Different heuristic algorithms have been developed and are good at different problem classes. Here a brief overview of the basic concepts are given for the most popular ordering schemes. For more complete descriptions of the orderings we refer to Duff et al. [14] and George and Liu [23]. Now the research interest has moved to find equivalent orderings that enhance desirable properties in the computed factors or reduce intermediate fill-in during the computation, see [34, 20].

The most popular general-purpose algorithm is the *minimum degree* ordering or Tinney scheme 2, by Tinney and Walker [66]. This is a special case of the unsymmetric Markowitz algorithm [47]. The minimum degree algorithm uses a local minimizing aspect. In each elimination step the pivot is chosen to generate the least fill-in in the current step. The name minimum degree relates to the interpretation of the effect on the graph from a symmetric matrix. In each step the node with the least number of edges is eliminated. This scheme has been very

successful and it is easy to see that for a graph without any cycles, a tree, this method generates no fill-in. The minimum degree algorithm is a local strategy and there are examples where the global fill-in is not minimized with this scheme. There has been a lot of effort to find good tie-break strategies, i.e, to determine which pivot to choose if several has the same degree, and to perform a fast approximate updating of the current elimination graph [1, 34, 40]. A survey of the minimum degree algorithm and recent ideas for speed improvement is given by George and Liu [24].

If is often natural to order a matrix in a way that gives a small bandwidth. The Cholesky factor R of a positive definite $n \times n$ symmetric matrix, $C = R^T R$, has the property that R has no element outside the band structure of C. If the bandwidth of C is minimized the bandwidth of R will also become small as well. Cuthill and McKee purposed an algorithm in 1963 which minimized the bandwidth locally. The Cuthill-McKee algorithm can be expressed in the following way. Choose a starting node, label this with one and continue to number the neighboring nodes to the first. Repeat this procedure with the second node and number only nodes that do not have a number already. A renumbering of the nodes corresponds to a symmetric permutation of a symmetric matrix.

A minimum bandwidth ordering corresponds to collecting the dependencies to the current pivot element as close as possible. In 1971 George found that if the Cuthill-McKee ordering was reversed the fill-in usually decreased significantly. This procedure is called the *Reverse Cuthill-McKee* ordering.

The *nested dissection* ordering tries to remove a set of nodes from the graph of a symmetric matrix in order of obtain two or more disconnected graphs. The set of the removed nodes is called a separator set and should be made as small as possible. The disconnected graphs can themselves be divided by new separator sets to any depth. It is also advantageous if the disconnected graphs are roughly of the same size. The structure after the first separation will be,

$$PAP^T = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix},$$

$A_{i,j}$, $i \neq j$ corresponds to the separator set. Each block A_{ii} can be treated separately and the blocks A_{ij} are updated and eliminated in the end. If the separator sets are small the blocks $A_{i,j}$ are also small. The method of nested dissection is especially successful on matrices from finite element problems and geodetic problems. Gilbert showed that this ordering was optimal in the order of magnitude for a matrix associated with an $n \times n$ rectangular grid [25, 23].

3.4 Sparse factorizations

The basic tools for computing the least squares problem are the QR and Cholesky factorizations. These factorizations have an advantage to Gaussian elimination of a linear $n \times n$ system. A permutation of A can be computed in advance without any regard to numerical stability provided A has full column rank. After the

permutation is chosen a symbolic factorization is made to predict and allocate storage for the R factor. The numerical factorization is then carried out with a static memory structure of R.

The factorization process of a matrix A can be split into three different steps:

1. Ordering: compute an ordering P so that $PA^TAP^T = R^TR$ produces a sparse R factor.

2. Symbolic factorization: compute the structure of R symbolically and set up the data structure associated with nonzero elements of R.

3. Numerical factorization: compute the numerical factorization R.

The symbolic factorization phase can be as expensive as the numerical factorization, see [49]. Therefore, if several factorizations are carried out with different matrices sharing the same structure, much can be gained by reusing the symbolic information. We will later see that the matrices generated by the interior-point method, share the same symbolic structure.

3.4.1 Sparse QR factorization

The basic tool for computing a backward stable solution to the least squares problem is the QR factorization. The currently most effective method for computing the sparse QR factorization is the multi-frontal QR algorithm, see [16, 39, 49, 64]. The multi-frontal algorithm rearranges the elements in A into small dense submatrices, *frontal matrices*, which can be handled more efficiently on vector computers and parallel computers. The basic concept is to create an *elimination tree* of the matrix A, which captures the dependencies between different rows. For a description of the elimination tree and its use in sparse factorizations we refer to Liu [40].

Selected rows from A can, with the information in the elimination tree, be packed into smaller dense frontal matrices. Each frontal matrix are completely reduced to upper triangular form by a dense QR factorization. Now, depending on the elimination tree, at least one row of the triangular matrix is moved to the final R. The remaining rows are put onto a stack for later usage in the frontal matrix corresponding to the parent node in the elimination tree. The different branches in the elimination tree can be processed independently and this can be used for a coarse grain parallelism.

We have used the semi-normal equations or solved the least squares problem with the Cholesky factorization, because in the interior-point the right hand side is not always known in advance. An alternative would be to compute the Q factor or at least a representation allowing us to apply Q^T on a vector. However, the method for computing the Q factor in the programming environment we have used is not satisfactory. The implementation in MATLAB computes Q by applying orthogonal transformations on the matrix (AI), where I is the $m \times m$ identity matrix. This is not an efficient way of computing Q and should be avoided. Further, even if the Q factor would be computed in an efficient way,

Table 3.2: The number of nonzero elements for different representations of Q.

Matrix	nnz(A)	nnz(R)	nnz(Q)	density Q	nnz(Y)	density Y
1	1557	1924	61409	0.64	3405	0.03
5	1916	2898	473426	0.52	12257	0.01
6	4732	2580	517074	0.48	11033	0.01
12	8758	7649	1674022	0.49	27421	0.008

and stored by its Householder vectors, it would still not be efficient. Gilbert, Ng and Peyton [28] analyzed the nonzero count for the Q matrix and matrix consisting of the Householder vectors, H. The result they obtained for a certain class of matrices with so called $\sqrt{n}$-separable graphs is the following, for a $n \times m$ matrix A, the nonzero count of H satisfies the following nonzero count, $nnz(H) = \mathcal{O}(n \log n + (m-n)\sqrt{n})$. This compared with the nonzero count of Q, nnz(Q) $= \mathcal{O}(m\sqrt{n})$ is good when $m \approx n$ but if we have a more rectangular matrix A the difference will not be as satisfactory. In Table 3.2 we compare the number of nonzero element in the Q factor obtain from MATLAB compared to the initial number of nonzero elements in A and in the R factor. This clearly shows how inefficient this approach is. We have used the minimum degree ordering in all cases.

In the multi-frontal setting we do not work directly on the matrix A, instead we collect certain rows and form frontal matrices. The natural way here to store a representation of Q would be to store the Householder transformations from each frontal matrix, Y_i. The information in the matrices Y_i together with the elimination tree is all that is needed to apply Q or Q^T to a vector. Lu and Barlow [43] investigated the nonzero count, nnz(Y) $= \sum_i$(nnz(Y_i)), from the same class of matrices that Gilbert, et al. considered. They obtained the following bound on the number of nonzeros by storing the Householder vectors from the frontal matrices, nnz(Y) $= \mathcal{O}(n \log n)$. In last two column of Table 3.2 we have used a modified version of `sqr`, by Matstoms [48], to compute nnz(Y). As could be expected we see a dramatic reduction of the storage needed for a representation of Q. The number of nonzero for the Q matrix obtained from MATLAB is unnecessary large, only the n first columns are needed in general.

3.4.2 Sparse Cholesky factorization

Since the R factor from the Cholesky decomposition mathematically is the same as from the QR factorization, except some sign differences, the symbolic step in the Cholesky factorization is much the same as for the QR method. Techniques to merge several columns with the "same" structure into larger *super nodes* are important to gain efficiency when computing the Cholesky factorization, see [41, 27]. Until recent time it has been common to use the given ordering and compute the Cholesky factor straight away. Now some research is done to adjust the ordering; so as to get equivalent orderings in regard to nonzero elements in R, but with more desirable properties in R.

4 Test problems for the BLS problem

Here we will describe the different test problems used in the comparison of the different methods. We have not used any randomly generated matrices with a randomly generated sparsity pattern due to the fact that these have very special properties and are not representative. Here the matrices used will be presented with their properties. Then the non trivial problem of generating a test problem with a known solution will be discussed. The method for generation of the test problems presented here is a small generalization of the method in Portugal et. al. [62].

4.1 Sparse test matrices

The matrices used to create the test problems come from three different sets. Each of the matrices is rectangular, with the number of rows greater than the number of columns, $A \in I\!R^{m\times n}$, $m > n$, where A has full column rank. The condition number, $\kappa(A)$, for the matrices will be given except for the really large matrices. There a lower estimation of the condition number in 1-norm is given instead. The LAPACK estimator (in MATLAB `condest`) have been used to compute the lower bound. For details of the estimator, see Higham [35, Chapter 14].

The first set is a subset from the much used Harwell-Boeing test matrix collection [5] Duff, Grimes and Lewis [15]. Table 4.1 summarizes some different properties of the Harwell-Boeing matrices. In the matrices ABBnnn and ASHnnn, the original data is replaced elementwise with uniformly distributed random numbers in the interval $[-1, 1]$. The matrices WELLnnnn are provided by M. A. Saunders and arises from gravity-meter observations. ILLCnnnn has the same structure as WELLnnnn but is ill-conditioned. The matrices ARTFnnnn are artificially constructed and are not included in the Harwell-Boeing collection set. They are built up in the following way

$$A = \begin{pmatrix} \boxed{\text{WELLnnnn}} \\ \boxed{\text{O}}\ \boxed{\text{ASHnnn}} \end{pmatrix}. \tag{4.1}$$

All these matrices are from the late 70's and are quite small compared to large sparse problems solved today. This set of matrices is used here because the Harwell-Boeing collection has been previously used in several articles see George, Heath and Ng [22], Heggernes and Matstoms [34], Lu and Barlow [43] and Matstoms [49, 50].

The second set, listed in Table 4.2, are some matrices from animal breeding science. Several parameters, for example, growth factor, final weight and quality of meat was measured. The model included several different environmental and generic effects to describe the evolution of the animal. The interest was to maximize economical value by selecting good animals for breeding. This

[5] The Harwell-Boeing collection and many other sparse matrices can be obtained from Matrix Market, URL:`http://math.nist.gov/MatrixMarket/`

Table 4.1: A subset of the Harwell-Boeing collection; nnz denotes the number of nonzero elements in the matrix and $Cond$ is the condition number in 2-norm.

No.	Matrix	m	n	nnz	$Cond$	Description
1	ABB313	313	176	1557	$1.5 \cdot 10^1$	Survey of Sudan
2	ASH219	219	85	438	$7.7 \cdot 10^0$	Surv. of UK, Holland
3	ASH331	331	104	662	$5.2 \cdot 10^0$	Surv. of UK, Holland
4	ASH608	608	188	1216	$6.3 \cdot 10^0$	Surv. of UK, Holland
5	ASH958	958	292	1916	$6.9 \cdot 10^0$	Surv. of UK, Holland
6	WELL1033	1033	320	4732	$1.7 \cdot 10^2$	Gravity-meter obs.
7	ILLC1033	1033	320	4732	$1.9 \cdot 10^4$	Gravity-meter obs.
8	ARTF1252	1252	320	5170	$3.6 \cdot 10^1$	8 + 2 (artificial)
9	ARTF1364	1364	320	5394	$3.6 \cdot 10^1$	8 + 1 (artificial)
10	ARTF1641	1641	320	5948	$4.1 \cdot 10^0$	8 + 4 (artificial)
11	ARTF1991	1991	320	6648	$3.9 \cdot 10^0$	8 + 5 (artificial)
12	WELL1850	1850	712	8758	$1.1 \cdot 10^2$	Gravity-meter obs.
13	ILLC1850	1850	712	8758	$1.4 \cdot 10^3$	Gravity-meter obs.
14	ARTF2808	2808	712	10674	$2.8 \cdot 10^1$	12 + 5 (artificial)

particular set has been used for pig breeding in Switzerland and was supplied by A. Hofer. A more detailed description can be found in [49, Appendix A].

Table 4.2: Matrices from breeding science. nnz denotes the number of nonzero elements in the matrix and $Cond$ is the condition number in 2-norm.

No.	Matrix	m	n	nnz	$Cond$	Description
15	SBREED	3140	1987	8506	$5.9 \cdot 10^2$	Breeding prob.
16	IBREED	9397	6118	24997	$9.1 \cdot 10^2$	Breeding prob.
17	LBREED	28254	17263	75002	$> 2.1 \cdot 10^3$	Breeding prob.

The third set of matrices arises in the natural factor formulation of finite element methods and has been used as least squares test problems by George et al. [22]. This problem has also served as a model problem by Lu and Barlow [43] when investigating the lower bound of the nonzero count in the Householder factors from the multi-frontal QR, see Section 3.4.1. Each grid consists of $(k-1)^2$ small squares. To each of the k^2 grid points a variable is associated and to each small square there are four observations involving the four corner variables. This leads to an overdetermined system of equations with $m = 4 \cdot (n-1)^2$ rows and $n = k^2$ columns. The nonzero entries in the matrix are uniformly distributed random numbers in the interval [0, 1] see Table 4.3.

Table 4.3: Matrices NFAC arises from k by k grid in a FEM calculation; nnz denotes the number of nonzero elements in the matrix and $Cond$ is the condition number in 2-norm.

No.	Matrix	m	n	nnz	$Cond$	Description
18	NFAC10	324	100	1296	$1.1 \cdot 10^1$	FEM 10x10 grid
19	NFAC20	1444	400	5776	$1.4 \cdot 10^1$	FEM 20x20 grid
20	NFAC30	3364	900	13456	$1.3 \cdot 10^1$	FEM 30x30 grid
21	NFAC40	6084	1600	24336	$1.3 \cdot 10^1$	FEM 40x40 grid
22	NFAC50	9604	2500	38416	$1.2 \cdot 10^1$	FEM 50x50 grid
23	NFAC60	13924	3600	55696	$1.2 \cdot 10^1$	FEM 60x60 grid
24	NFAC70	19044	4900	76176	$1.2 \cdot 10^1$	FEM 70x70 grid
25	NFAC80	24964	6400	99856	$> 1.8 \cdot 10^1$	FEM 80x80 grid
26	NFAC90	31684	8100	126736	$> 1.9 \cdot 10^1$	FEM 90x90 grid

4.2 Generation of test problems

If we want to be able to determine the accuracy we have obtained in the solution from the methods we have studied, we want to have numerical test examples with a solution known with an accuracy equal to machine precision. We also want to be able to change the characteristics of the test problem for a given matrix A, for example the degeneracy of the solution.

For unconstrained least squares problem we can create a consistent problem with a known solution x by letting $b = Ax$. By adding a vector $r \in \mathcal{N}(A)$ a problem with a nonzero residual vector is obtained.

The same construction cannot be done with constrained least squares problem, because these problems can not be separated into a constrained and an unconstrained part with no relationship between them. A degenerated problem could be created by the method above, if we let the upper or lower bound be equal to the solution x_i for some i. However, this kind of problem is not in general of interest.

For the general quadratic programming problem, $q(x) = \frac{1}{2}x^T Ax - x^T g$, with only upper and lower bounds the standard way to generate a test problem is to set the solution to x. To guarantee that x is the solution to the quadratic programming problem we determine the vector $g = Ax - r$, where r is the Lagrangian multipliers. The components of r are determined as follows,

$$\begin{aligned} r_i &= 0 \quad i \in \mathcal{F}, \\ r_i &> 0 \quad i \in \mathcal{B}_\mathcal{L}, \\ r_i &< 0 \quad i \in \mathcal{B}_\mathcal{U}. \end{aligned}$$

The amount of degeneracy can be specified by choosing the magnitude of r_i, see Moré and Toraldo [56] and Coleman and Hulbert [9].

In least squares problems we have the term $-x^T A^T b$ instead of $-x^T g$. Therefore we have to solve the underdetermined problem $A^T b = g$ to obtain the vector

b. This has an infinite number of solutions, so we are satisfied by the minimum norm solution given by the normal equations of the second kind.

We now describe a way to generate the test problems so that, we have the possibility to predetermine the following different characteristics of the test problem. We can chose the solution x, the number of variables bounded to upper and lower bounds, the number of constraints degenerate at the solution, and the size of the Lagrange multipliers to the active constraints.

For simplicity, consider the problem,

$$\begin{aligned} &\min_x ||Ax - b||_2, \\ &\text{s. t.} \quad 0 \leq x \leq u. \end{aligned}$$

A problem with arbitrary upper and lower bounds can then be obtained by a simple translation of this problem.

The components x_i of the solution to this problem belong in one of the following five sets; the free variables $\mathcal{F}$, the bounded variables with nonzero Lagrange multiplier $\mathcal{L}_B$ and $\mathcal{U}_B$, and the bounded variables where the associated Lagrange multiplier is zero, $\mathcal{L}_D$ and $\mathcal{U}_D$. These last two sets determine the degenerate constraints.

Let v and y denote the Lagrange multiplier associated with the lower and upper bounds, The value of the Lagrange multipliers is constrained by the parameter α. The sets are characterized by the following:

i)	$x_i = 0$	$v_i = 0$	$i \in \mathcal{L}_D$
ii)	$x_i = 0$	$v_i > 0$	$i \in \mathcal{L}_B$
iii)	$x_i = u_i$	$y_i = 0$	$i \in \mathcal{U}_D$
iv)	$x_i = u_i$	$y_i > 0$	$i \in \mathcal{U}_B$
v)	$l_1 < x_i < u_i$	$v_i = y_i = 0$	$i \in \mathcal{F}$

Let $w = v - y$ and observe that v_i and y_i can never be greater than zero at the same time. Let $\mathcal{L} = \mathcal{L}_B \cup \mathcal{L}_D$ and $\mathcal{U}$ in the same way. Let $x_i = 0$ for all $i \in \mathcal{L}$, $x_i = u_i$ for all $i \in \mathcal{U}$ and generate uniform random numbers to all remaining x_i, $i \in \mathcal{F}$. Let $w_i = 0$ for all $i \in \mathcal{L}_D \cup \mathcal{U}_D \cup \mathcal{F}$ and generate proper random numbers so the conditions above are satisfied. The construction of the right hand side b can then be done in the following way. From the feasibility condition we have the following relation:

$$w = A^T(Ax - b). \tag{4.2}$$

Partition A into the following parts $A = (A_{\mathcal{L}}\ A_{\mathcal{U}}\ A_{\mathcal{F}})$, where $A_{\mathcal{L}}$ are the column corresponding to the variables x_i, $i \in \mathcal{L}$ etc. If the characteristics of the sets are used with 4.2, we obtain with this partitioning of A the following:

$$A^T b = \begin{pmatrix} A_{\mathcal{L}_D}^T q \\ A_{\mathcal{L}_B}^T q - w_{\mathcal{L}} \\ A_{\mathcal{U}_D}^T q \\ A_{\mathcal{U}_B}^T q - w_{\mathcal{F}} \\ A_{\mathcal{F}}^T x_{\mathcal{F}} \end{pmatrix} = c, \qquad q = A_{\mathcal{U}} x_{\mathcal{U}} + A_{\mathcal{F}} x_{\mathcal{F}}. \tag{4.3}$$

This forms an under-determined system which can have infinite number of solutions. The solution which gives the smallest residual in 2-norm is given by,

$$\min_b ||b - Ax||_2, \qquad A^T b = c.$$

By using Observation 2.4, this can be rewritten as the equivalent problem,

$$\min_b ||b||_2, \qquad A^T b = c.$$

If this system is consistent and of rank n, the solution is given by the normal equations of the second kind, see Björck [5],

$$A^T A z = c, \qquad b = Az.$$

By computing R in a sparse QR decomposition of A, we obtain the solution b from the corrected semi normal equations (SNE),

Since SNE is not backward stable, see Section 2.3, we use iterative improvement to obtain an accurate solution. Higher precision arithmetic (quadruple precision, 128 bits) was used to calculate the residual in the iterative refinement. In most cases the iterative refinement converges quickly. Björck and Paige [6] have shown that if $\mathbf{u}\kappa^2 < 1$, only one step of iterative refinement can give an acceptable-error solution. This algorithm was implemented in the

---

ALGORITHM **4.1 (Generation of test problem)**

Choose the vector u and parameter α
Partition the set $[1,\ 2, \ldots,\ n] = \mathcal{F} \cup \mathcal{L}_B \cup \mathcal{L}_D \cup \mathcal{U}_B \cup \mathcal{U}_D$
$x_{\mathcal{L}} = 0$, $x_{\mathcal{U}} = u_{\mathcal{U}}$, $x_{\mathcal{F}} = rand([\varepsilon_1,\ u_i - \varepsilon_1])$
$w_{(\mathcal{L}_D \cup \mathcal{U}_D \cup \mathcal{F})} = 0$
$w_{\mathcal{L}_B} = rand([\varepsilon_2,\ \alpha])$, $w_{\mathcal{U}_B} = rand([-\alpha,\ -\varepsilon_2])$
c is given by (4.3)
Compute the QR factorization of A
Solve SNE: $R^T R z = c$, $b = Az$
***do** iterative refinement three times*
    Compute in higher precision $s = c - A^T b$
    Solve $R^T R dy = -s$
    $b = c - Ady$
end

MATLAB routine `makexb`.

Algorithm 4.1 was used with each test matrix to create two different kinds of test problem, type A and B;

- **Type A** are nondegenerate and have variables $\frac{1}{4}$ bounded to the lower bound, $\frac{1}{4}$ bounded to the upper bound and $\frac{1}{2}$ free.
- **Type B** are degenerate and have $\frac{1}{2}$ free variables; the other are equally divided between the remaining four sets.

The upper limit for all problems was set to 10, the lower limit to 0 and the maximum value of the Lagrange multipliers was set to 10. All random values are uniformly distributed in the interval. The MATLAB function `rand` was used to create the random numbers. Quadruple precision (128 bits) was used in the iterative refinement phase to compute an accurate right hand side b.

5 Active set methods

In the well known simplex method for linear programming (LP) problems, the solution is found by trying to identify the active constraints at the solution, the *active set* of constraints. In the simplex algorithm this set is obtained by walking from vertex to vertex of the feasible domain towards the solution. In the methods called active set methods a temporary working set of constraints are treated as equality constraints and a new solution is computed from the smaller unconstrained problem. In each iteration constraints are either added or removed from the active set until the optimum of the constrained problem is found. In the simplex algorithm only one constraint, in general, is added/removed from the active set in each iteration. This can be inefficient when the number of constrains are large. If only one constraint is moved in each iteration a lower limit for the number of iterations before the method can converge is the difference between the number of active constraints in at starting point and at the solution. A block method where several constraints are moved in each iteration can therefore be more efficient.

We will in this section present several block methods that work well for large problems. Convergence proofs of the single pivot method and a certain block method will be given. If the matrix A is rank deficient a unique solution may not exist, and a method for computing the minimum norm solution for such problems will be presented.

For the active set methods given here, upper and/or lower bounds may not be present. This can be formulated in the following way:

$$\begin{array}{lll} & \min_x \|Ax-b\|_2, & \\ \text{subject to} & l_i \le x_i \le u_i, & i = 1, ..., k, \\ & l_i \le x_i, & i = k+1, ..., l \le n, \end{array} \tag{5.1}$$

where A is large and sparse. Problems with an upper bound on x_i but no lower bound can be reformulated as (5.1) by changing sign of the corresponding variable x_i and column of A and defining the lower bound l_i as $-u_i$.

5.1 Previous work

Several papers have been written on active set methods for definite and indefinite QP problems with simple bounds. In Lawson and Hanson, [38], an active set method NNLS is used for dense nonnegative LS problems ($x > 0$). This algorithm is currently used in the routine with the same name in MATLAB V5. NNLS is a single pivoting method and may not be suitable for large scale problems. A sparse version of NNLS was developed by Oreborn [60]. An extension of this method to sparse least squares problem with lower and upper bounds has been proposed by Björck [4]. The emphasis of this work is on how to efficiently update the QR factorization of the active columns of A.

Lötstedt [42] takes a similar approach but uses a preconditioned cg method to solve the linear system of equations. A method for finding the solution with

minimal norm in the rank deficient case is also discussed here. The method was developed to solve a sequence of problems with slowly varying data found in rigid mechanics.

For a treatment of general QP problems with simple bounds, see Coleman and Hulbert [9], or Gill and Murray [29] for general dense quadratic programming problems. Moré and Toraldo [56] suggest a combined active set and gradient projection algorithm for quadratic programming problems with bounds. In the article the projected gradient is used for computing the search directions and a proof of convergence is presented. They report a decrease of iterations by at least a factor 10, but the cost of solving the system of equation will increase. One of the algorithms we develop in the following, has a close relationship with the Algorithm BCQP suggested in the paper above.

In a recent paper by Dostál [13] an alternative to the projected gradient is proposed. He shows the convergence in a finite number of steps for an algorithm driven by proportioning with conjugate gradients.

The block method was first suggested for a certain class of functions by Kostreva [37] for the linear complementarity problem. A comparison of the single and block pivoting method and interior-point methods for the nonnegative least squares problem has been given in [62].

5.2 Characterization of the solution and optimality conditions

We will now show how we can obtain the solution to (5.1) if information about the bounded and the free variables is known. The index set of an arbitrary point x in the feasible domain can be divided into two different subsets, $[1 \dots n] = \mathcal{F} \cup \mathcal{B}$. The set $\mathcal{B}$ is the active set and consists of the index set of variables at the limits,

$$\mathcal{B}(x) = \{i : x_i = \delta_i, \delta_i = l_i, \text{ or } \delta_i = u_i\}.$$

The remaining indexes of the variables form the free set,

$$\mathcal{F}(x) = \{i : l_i < x_i < u_i\}.$$

If the partition of the sets at optimal point to (5.1) is known we can use this partition to obtain a problem with equality constraints instead of upper and lower bounds. This problem is now solved by,

$$\min_{x_{\mathcal{F}}} ||A_{\mathcal{F}} x_{\mathcal{F}} - (b - A_{\mathcal{B}} x_{\mathcal{B}})||_2, \tag{5.2}$$

and we have the solution to (5.1).

By using the sets defined above we can formulate the first order optimality conditions from Theorem 1.5. Let the sets $\mathcal{F}$ and $\mathcal{B}$ be defined as above and divide the set $\mathcal{B}$ into two sets corresponding to the variables bounded by the upper constraint, $\mathcal{B}_{\mathcal{U}}$, and by the lower constraint $\mathcal{B}_{\mathcal{L}}$. The first order optimality

conditions can then be expressed in the following way,

$$\begin{aligned} y_{\mathcal{B}_U} &= A^T_{\mathcal{B}_U}(b - Ax), && (5.3)\\ -v_{\mathcal{B}_L} &= A^T_{\mathcal{B}_L}(b - Ax), && (5.4)\\ y_{\mathcal{B}_L} = y_{\mathcal{F}} &= v_{\mathcal{B}_U} = v_{\mathcal{F}} = 0, \\ v_i(x_i - l_i) &= 0,\ \forall i, \\ y_i(u_i - x_i) &= 0,\ \forall i, \\ v, y &\geq 0, \end{aligned}$$

where y and $-v$ are the Lagrangian multipliers to the upper and lower constraints. If the optimum is not found when we have evaluated the multipliers above, we can reduce the objective function further by releasing the index of a bound variable with a negative multiplier to the set $\mathcal{F}$. The optimum is found when $x_{\mathcal{F}}$ is a stationary point and y, v corresponding to the variables in $\mathcal{B}$ have positive signs.

If the matrix A is rank deficient, $\text{rank}(A) = r < n$ then the solution is not unique if $\mathcal{N}(A) \cap W \neq \emptyset$, W is base for all feasible directions at x. The solution with minimum norm can then be found by solving $\min ||x + p||_2$, $p \in \mathcal{N}(A) \cap W$ subject to the constraints, see section 5.5

5.3 Single pivoting methods

The usual way to implement the active set methods as done, e.g, in the MATLAB algorithm `NNLS` [38, p. 161] is to solve the unconstrained problem in the free variables in each iteration. Let $x^{(n)}$ be the solution after n iterations with the single pivoting algorithm. To compute the next iterate we solve,

$$\min_p ||A(x^{(n)} + Z_{\mathcal{F}_n} p) - b||_2, \quad p \in I\!R^l,\ l = |\mathcal{F}_n|, \tag{5.5}$$

where $Z_{\mathcal{F}_n} \in I\!R^{l \times n}$ is a matrix which consists of the columns e_i, $i \in \mathcal{F}_n$, where e_i is the i:th column from the identity matrix. If $p^{(n)}$ solves (5.5) we set $x^{(n+1)} = x^{(n)} + \alpha^{(n)} Z_{\mathcal{F}_n} p^{(n)}$, with $\alpha = \max\{\alpha : l \leq x^{(n)} + \alpha^{(n)} Z_{\mathcal{F}_n} p^{(n)} \leq u\}$ as shown in Figure 5.1. If $\alpha < 1$ we can not reach the global minimizer to (5.5) without violating some constraints, and the indices of the variables at the boundary are moved from the set $\mathcal{F}_n$ to $\mathcal{B}_n$. In general, the single pivoting methods move only one variable between the subsets in each iteration. More than one variable can be bound if more than one constraint are satisfied exactly for the same value of α. This is repeated until we find a global minimizer to (5.5). Then $x^{(n+1)}$ is checked for optimality by calculating the Lagrange multipliers. If there is a negative Lagrange multiplier the objective function can be reduced by allowing the corresponding constraint to become free in the next iteration. If more than one variable is not optimal, there are several ways to determine which one that should be moved. The simplest way to choose the multiplier is to take the one with lowest index, Murty's method [62],

$$\min i,\ \forall i : v_i \text{ or } y_i < 0.$$

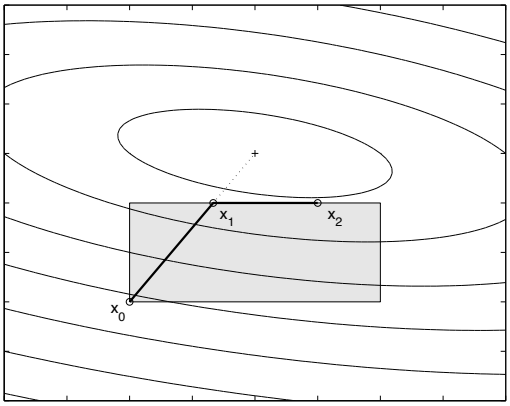

Figure 5.1: Iteration to convergence with the single pivoting strategy.

Alternatively one can choose the Lagrange multiplier with the largest absolute value (used in `NNLS`),

$$\min_{i,j}(v_i\ y_j),\ i \in \mathcal{B}_{\mathcal{L}} \text{ and } j \in \mathcal{B}_{\mathcal{U}}.$$

In other words, we choose the variable with which we can obtain the largest reduction of the objective function per unit step. This work well in practice, but one disadvantage is that it is not invariant to scalings of the constraints.

When we have simple bounds in form of box constraints it is easy to normalize the constraints so that this quantity becomes invariant by solving

$$\min_i \left(\frac{v_i}{u_i - l_i},\ \frac{y_j}{u_j - l_j} \right),\ i \in \mathcal{B}_{\mathcal{L}} \text{ and } j \in \mathcal{B}_{\mathcal{U}}. \tag{5.6}$$

The main weakness of the single pivoting active set method is that it takes at least $||\mathcal{B}^*| - |\mathcal{B}_0||$ number of iterations to converge, where $|\mathcal{B}^*|$ and $|\mathcal{B}_0|$ are the number of active constraints in the solution, and the number of active constraints initially respectively. This number can be very large for large-scale problem, see Table 5.1.

There are several ways to choose the initial active set $\mathcal{B}$, in the algorithm `NNLS`. In Portugal et. al. [62], all constraints are taken to be active initially, $\mathcal{B}_0 = [1, \ldots, n]$. This is not a good choice if there are few bounded variables at the solution, and in particular if no constraint is active. As in Oreborn [60] we have chosen the starting point as $P[x^*]$, where x^* is the solution to the unconstrained problem and $P[\cdot]$ is the orthogonal projection onto the feasible domain. In the case of simple bounds this projection is very easy to compute, in $\mathcal{O}$(n) operations. We have,

$$P[x] = \text{mid}(l, u, x),$$

where $\mathrm{mid}(l, u, x)$ is the vector whose i-th component is the median of the set l_i, u_i, x_i. If this starting point is chosen the algorithm will converge after the first iteration if no constraint is active at the solution. In our experience the number of iterations has decreased significantly by choosing this starting point.

ALGORITHM **5.1 (SINGLE. Single pivot active set.)**

Choose a column permutation P and permute A, l and u.
Compute $\begin{pmatrix} R \\ 0 \end{pmatrix} = Q^T A$ *and* $\begin{pmatrix} c \\ d \end{pmatrix} = Q^T b$ *by QR factorization.*
Solve unconstrained problem $x^* : \|Rx^* - c\|_2 = \min$.
$x^{(0)} = P[x^*]$; *n=0*
Calculate Lagrange multipliers y and v by (5.3)-(5.4).
Initiate the index sets $\mathcal{F}$, $\mathcal{L}$ and $\mathcal{U}$ accordingly.
while *not optimal point.*
  while *not constrained point.*
    Update R: $Q_n^T(R_{\mathcal{F}_n}\ R_{\mathcal{B}_n} \mid c) = \left(\begin{array}{cc|c} U_n & S_n & d_n \\ 0 & V_n & e_n \end{array}\right)$
    Solve $U_n \overline{x}_{\mathcal{F}_n} = d_n - S_n x_{\mathcal{B}_n}^{(n)}$
    Find $\mathcal{L} : \overline{x}_{\mathcal{F}_n} \le l_{\mathcal{F}_n}$
    Find $\mathcal{U} : u_{\mathcal{F}_n} \le \overline{x}_{\mathcal{F}_n}$
    Calculate $[\alpha, \mathcal{I}] = \min\left(\frac{\overline{x}_{\mathcal{L}} \perp l_{\mathcal{L}}}{\overline{x}_{\mathcal{L}} \perp x_{\mathcal{L}}}, \frac{u_{\mathcal{U}} \perp \overline{x}_{\mathcal{U}}}{x_{\mathcal{U}} \perp \overline{x}_{\mathcal{U}}}\right)$
    $x^{(n+1)} = x^{(n)} + \alpha Z_{\mathcal{F}_n}(\overline{x}_{\mathcal{F}_n} - x_{\mathcal{F}_n}^{(n)})$
    $F_{n+1} = \mathcal{F}_n \setminus (\mathcal{L} \cup \mathcal{U})_{\mathcal{I}}$
    $\mathcal{B}_{n+1} = \mathcal{B}_n \cup (\mathcal{L} \cup \mathcal{U})_{\mathcal{I}}$
    $n = n + 1$
  end while
  Calculate Lagrange multipliers y and v.
  Use (5.6) to relase a variable from $\mathcal{L}$ or $\mathcal{U}$ to $\mathcal{F}_{n+1}$.
end while

Initially we compute the QR factorization of the matrix A to solve the first unconstrained problem. The main work is to update the QR factorization in each step, therefore it is important that this procedure is as efficient as possible, see Björck [4]. It is easy to show that all $R_{\mathcal{F}_n}$ factors can then be stored in the the initial R factor if the columns always keep their relative position to each other. A static data structure can be predicted from the initial factorization of A and kept throughout the iterations.

Portugal et. al. [62] start with all variables at their bound, if this is done none of the important features mentioned above will be available.

5.3.1 Convergence of single pivoting active set

Convergence in a finite number of steps for the single pivoting method was proved for the NNLS problem by Lawson and Hanson [38]. Lötstedt in [42]

outlined a proof for the BLS problem. A more general proof for the convergence of the single pivoting algorithm to a local minimum for a quadratic function with box constraints was given by Coleman and Hulbert [9]. The proof below follows the proof by Coleman and Hulbert. A small generalization is made in the choice of how to reintroduce a bound variable into the free set.

Define the corresponding quadratic function to the least squares problem by,

$$q(x) = \tfrac{1}{2}x^T A^T A x - x^T A^T b.$$

The gradient to $q(x)$ is $g(x) = A^T(Ax - b)$. Let $x_{\mathcal{F}} = Z_{\mathcal{F}}^T x$, $A_{\mathcal{F}} = Z_{\mathcal{F}}^T A$, and define the *reduced* gradient by $g_{\mathcal{F}} = Z_{\mathcal{F}}^T g$. If $x_{\mathcal{F}}$ is a stationary point then

$$g_{\mathcal{F}} = Z_{\mathcal{F}}^T g = Z_{\mathcal{F}}^T A^T(Ax - b) = A_{\mathcal{F}}^T(A_{\mathcal{F}} x_{\mathcal{F}} - (b - A_{\mathcal{B}} x_{\mathcal{B}})) = 0.$$

LEMMA 5.1 *The innermost while loop terminates after a finite number of iterations.*

Proof: The set $\mathcal{F}$ is finite and in each iteration we move one element from the set. If this is not the case, the point is a stationary constrained point and the loop terminates. If the set $\mathcal{F}$ is empty we consider $x_{\mathcal{F}}$ a stationary constrained point. □

LEMMA 5.2 *The function value at successive constrained stationary points is monotonic decreasing.*

Proof: Let x be the current stationary constrained point with $\hat{\mathcal{F}}$ as the free set. Then $g_{\hat{\mathcal{F}}}(x_{\hat{\mathcal{F}}}) = 0$. We move some index set f from $\hat{\mathcal{B}}$ to $\hat{\mathcal{F}}$, let $\mathcal{F} = \hat{\mathcal{F}} \cup f$ and $\mathcal{B} = \hat{\mathcal{B}} \setminus f$. Then we have $g_{\mathcal{F}}(x_{\mathcal{F}}) = \begin{pmatrix} 0 \\ v \end{pmatrix}$, where $v_i \neq 0$, and v has the same length as f. If the step size α is equal to zero in the next iteration we move index from $\mathcal{F}$ to $\mathcal{B}$, this variable can not be the one of those we reintroduced into $\mathcal{F}$, because these still represent feasible descent directions into the domain. The maximum number of iterations with step length 0 can be as most $|\hat{\mathcal{F}}|$. Then we have to take a step with positive α.

If the step size α is greater than zero the following will happen. In the next iteration we solve $\min_{\overline{x}_{\mathcal{F}}} ||A_{\mathcal{F}}\overline{x}_{\mathcal{F}} - \overline{b}||_2$, $\overline{b} = (b - A_{\mathcal{B}} x_{\mathcal{B}})$. This solution satisfies the normal equations $A_{\mathcal{F}}^T A_{\mathcal{F}} \overline{x} = A_{\mathcal{F}}^T \overline{b}$ and if we define the direction $p = \overline{x}_{\mathcal{F}} - x_{\mathcal{F}}$, we get

$$\begin{aligned} A_{\mathcal{F}}^T A_{\mathcal{F}} p &= A_{\mathcal{F}}^T A_{\mathcal{F}}(\overline{x}_{\mathcal{F}} - x_{\mathcal{F}}) \\ &= A_{\mathcal{F}}^T \overline{b} - A_{\mathcal{F}}^T A_{\mathcal{F}} x_{\mathcal{F}} = -g_{\mathcal{F}}(x), \end{aligned}$$

so the vector p defines a descent direction to $q(x)$. If the matrix $A_{\mathcal{F}}$ has full

column rank we get

$$\begin{aligned} q(x+\alpha Zp) - q(x) &= \frac{1}{2}\alpha^2 p^T A_{\mathcal{F}}^T A_{\mathcal{F}} p + \alpha (A^T Ax - A^T b)^T Z_{\mathcal{F}} p \\ &= \frac{1}{2}\alpha^2 g_{\mathcal{F}}^T (A_{\mathcal{F}}^T A_{\mathcal{F}})^{\perp 1} g_{\mathcal{F}} + \alpha g^T Z_{\mathcal{F}} p \\ &= \frac{1}{2}\alpha^2 g_{\mathcal{F}}^T (A_{\mathcal{F}}^T A_{\mathcal{F}})^{\perp 1} g_{\mathcal{F}} - \alpha g_{\mathcal{F}}^T (A_{\mathcal{F}}^T A_{\mathcal{F}})^{\perp 1} g_{\mathcal{F}} \\ &= \left(\frac{1}{2}\alpha^2 - \alpha\right) g_{\mathcal{F}}^T (A_{\mathcal{F}}^T A_{\mathcal{F}})^{\perp 1} g_{\mathcal{F}} < 0. \end{aligned} \tag{5.7}$$

If the matrix $A_{\mathcal{F}}^T A_{\mathcal{F}}$ is positive definite then the inverse has the same property, and therefore (5.7) is strictly negative for all $\alpha \in (0,\ 2)$. Choose the largest $\alpha \leq 1$ such that $l \leq x + \alpha Zp \leq u$ then $q(x+\alpha Zp) < q(x)$. If A is rank deficient we have multiple solutions when we solve the least squares problem. The term $(A_{\mathcal{F}}^T A_{\mathcal{F}})^{\perp 1}$ will be replaced by the pseudo-inverse, $(A_{\mathcal{F}}^T A_{\mathcal{F}})^{\dagger}$. Then the solution $\overline{x}$ must be chosen so the vector p is not in the null space of $A^T A$. If this is done there will be reduction as before. If $\overline{x}$ can not be chosen in this way then x is the optimal solution. □

Now we put these lemmas together and prove the convergence of the single pivoting active set method.

THEOREM 5.3 *The single pivoting active set method converges in finite number of steps.*

Proof: The inner loop terminates in a finite number of steps, so what is left is to prove that the outer loop terminates in a finite number of steps. Each constrained stationary point $x^{(k)}$ is the global solution to $\min q(x)$ with $x_i = l_i$ or u_i, $i \in \mathcal{B}$, a equality constrained problem. By Lemma 5.2 the function value is strictly less for two following stationary points. Since there is only a finite number of equality constrained problems and $q(x^{(k)})$ decreases with k we can not cycle. The outer loop terminates in a finite number of steps. This point satisfies the KKT conditions and is therefore the global minimizer. □

We will see later that a block method can be constructed so that this kind of reasoning will prove the convergence of the block method.

5.4 Block pivoting

One of the first to propose the block pivoting strategy was Kostreva [37], who used it for solving the nonlinear complementarity problem for a special class of functions called P-functions. For a nondegenerate problem the convergence of the block algorithm is proved for this type of functions, but the proof breaks down in the degenerate case. In [62] another block method was proposed and applied to the sparse nonnegative constrained least squares problem. Algorithm 5.3 below is an extension of this algorithm to lower and upper bounds. The

different block algorithms will be compared and a proof of the convergence of Algorithm 5.4 will be presented.

In block methods we try to move more than one variable between the active and the free set in each iteration. We can do this by projecting the solution of the unconstrained problem onto the domain by a orthogonal projection, see Figure 5.2, instead of taking a point on the line between the new point and the old. However, there is no guaranty that the objective function will decrease at the new point and in some rare cases a cyclic behavior will emerge. Figure 5.3 shows an iteration with no decrease in the objective function (but no cycling will occur here). Because of this behavior a safeguard must be built into the algorithm to detect the cyclic behavior.

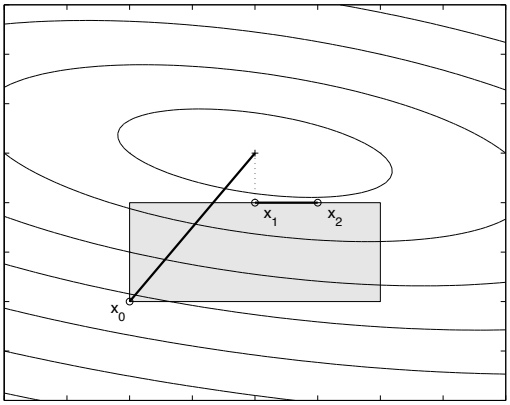

Figure 5.2: Iteration to convergence with the the block pivoting strategy in Algorithm 5.2.

In Algorithm 5.2, we have extended the single pivot algorithm by using projections when we determine the next iteration point. In each inner iteration several variables will be bound at their constraints and therefore it is not efficient to update the QR factorization in each step.

Since all variables in the active set methods always are feasible, we will denote an *infeasible* variable as a bounded variable with negative Lagrangian multiplier, or a free variable for which belongs to the solution to the unconstrained least squares problem (5.2) but do not belong in the feasible domain.

To ensure convergence a heuristic method for detecting cycling has been included. This heuristic counts the number of infeasible variables in each iteration and switches to a single pivot algorithm if no reduction was found in the last T iterations. The single pivot algorithm is used until a reduction of this quantity is obtained.

Portugal et. al. [62] have followed another approach for generalizing a single pivot strategy to a block method. They begin with the single pivoting method

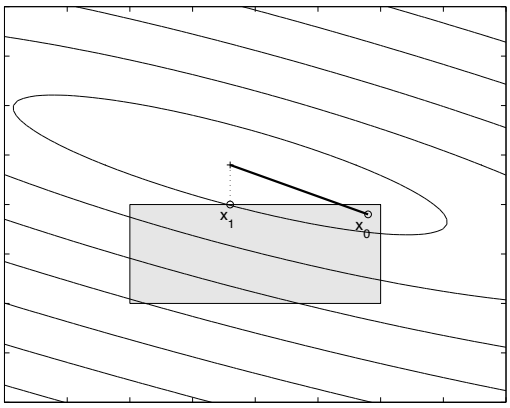

Figure 5.3: Block pivoting algorithm with no reduction of the objective function.

ALGORITHM **5.2 (BLOCK1. Block pivoting active set.)**

Choose a column permutation P and permute A, l and u.
Initiate the index sets $\mathcal{F}_0 = \{1 : n\}$ and $\mathcal{B}_0 = 0$.
Set $x^{(0)} = 0$
***while** not optimal point.*
  ***while** $x_{\mathcal{F}_n}$ not stationary point.*
    Solve $\min_{x_{\mathcal{F}_n}} ||A_{\mathcal{F}_n} x_{\mathcal{F}_n} - (b - A_{\mathcal{B}_n} x_{\mathcal{B}_n})||_2$
    Find $\mathcal{L} : x_{\mathcal{F}_n} \leq l_{\mathcal{F}_n}$
    Find $\mathcal{U} : u_{\mathcal{F}_n} \leq x_{\mathcal{F}_n}$
    $\mathcal{F}_{n+1} = \mathcal{F}_n \setminus (\mathcal{L} \cup \mathcal{U})$
    $\mathcal{B}_{n+1} = \mathcal{B}_n \cup \mathcal{L} \cup \mathcal{U}$
    $x^{(n+1)} = Z_{\mathcal{F}_n} P[x_{\mathcal{F}_n}] + Z_{\mathcal{B}_n} x_{\mathcal{B}_n}$
  end while
  Calculate Lagrange multipliers.
  Release infeasible variables from $\mathcal{B}$.
  ***if** no reduction of the number of infeasible variables has been made during the last T iterations switch to single pivoting mode until reduction made in the number of infeasible variables.*
end while

---

by Murty [58] and create a block variant by moving all infeasible variables x_i to $\mathcal{B}$. At the same time they release all variables corresponding to Lagrange multipliers with wrong sing. In the single pivoting method by Murty only one index is moved in each iteration either to $\mathcal{B}$ or from $\mathcal{B}$. This scheme has proven very effective for the test problems, but the algorithm may cycle. Hence as for the `BLOCK1` algorithm the safeguard to switch to a single pivot strategy must be included. It is shown later i Table 5.5, that this method can be quite ineffective if the algorithm switches over to the single pivoting mode.

ALGORITHM **5.3 (`BLOCK2`. Ext. of the BLOCK in Portugal et. al.,[62])**

Choose a column permutation P and permute A, l and u
Initiate the index sets $\mathcal{F} = \{1 : n\}$ and $\mathcal{B} = 0$
***while** not optimal point*
solve $\min_{x_{\mathcal{F}_n}} ||A_{\mathcal{F}_n} x_{\mathcal{F}_n} - (b - A_{\mathcal{B}_n} x_{\mathcal{B}_n})||_2$
Calculate Lagrange multipliers y, v
$\mathcal{H}_1 = \{i \in \mathcal{F} : x_i \leq l_i \text{ or } x_i \geq u_i\}$
$\mathcal{H}_2 = \{i \in \mathcal{B} : v_i \leq 0 \text{ or } y_i \leq 0\}$
$\mathcal{F} = (\mathcal{F} \cap \mathcal{H}_2) \setminus \mathcal{H}_1$
$\mathcal{B} = (\mathcal{B} \cap \mathcal{H}_1) \setminus \mathcal{H}_2$
$x^{(n+1)} = Z_{\mathcal{F}} P[x_{\mathcal{F}}] + Z_{\mathcal{B}} x_{\mathcal{B}}$
***if** no reduction of the number of infeasible variables has been made during the last T iterations switch to single pivoting mode until reduction made in the number of infeasible variables.*
end while

By modifying the first block algorithm, `BLOCK1`, we can ensure that Lemma 5.2 is true for the block algorithm and therefore the convergence proof holds, see Theorem 5.4. We know that we can obtain a reduction of $q(x^{(n)})$ by binding the first constraint we approach as in the single pivoting algorithm. However, since the `BLOCK2` algorithm is so efficient we want to bind as many constraints as possible and still obtain a reduction of the objective function. This could be achieved if we could compute,

$$\alpha = \max\{\alpha \mid q(P[x^{(n)} + \alpha p]) < q(x^{(n)})\},$$

but this would be very expensive. An approximation of this α could be obtained by first computing all α such that a new constraint is violated. This computation is cheap, $\mathcal{O}(n)$ and is already made in each step of the single pivot algorithm for finding the smallest α. When we have this quantity we can solve,

$$\begin{aligned} \alpha &= \max\{\alpha_i \mid q(P[x^{(n)} + \alpha_i p]) < q(x^{(n)})\}, \\ \alpha_i &= \left(\frac{\overline{x}_{\mathcal{L}}}{\overline{x}_{\mathcal{L}} - x_{\mathcal{L}}}, \frac{u_{\mathcal{U}} - \overline{x}_{\mathcal{U}}}{x_{\mathcal{U}} - \overline{x}_{\mathcal{U}}}, 1 \right). \end{aligned} \tag{5.8}$$

To be able to obtain the largest α_i we compute the quantity $||AP[x^{(n)} + \alpha_i p] - b||_2^2$ starting with the largest α_i in descending order until a reduction is obtained.

This is called a backtracking procedure by Dembo and Tulowitzski [12]. The main part of each computation is a matrix-vector multiplication, $\mathcal{O}(nnz)$, and is not too expensive compared to the factorization at each step. We can expect a decrease in efficacy if we have to compute q for many different α. Observe that a part of the computation of q is needed for the computation of the Lagrange multipliers y and v. Hence if we obtain a reduction with a full step, $\alpha = 1$ there is almost no extra cost for this computation.

ALGORITHM **5.4 (BLOCK3. Extension of algorithm BLOCK1.)**

Choose a column permutation P and permute A, l and u.
Initiate the index sets $\mathcal{F}_0 = \{1 : n\}$ and $\mathcal{B}_0 = 0$.
Set $x^{(0)} = 0$
***while** not optimal point.*
  ***while** $x_{\mathcal{F}_n}$ not constrained point.*
    solve $\min_{x_{\mathcal{F}_n}} ||A_{\mathcal{F}_n} x_{\mathcal{F}_n} - (b - A_{\mathcal{B}_n} x_{\mathcal{B}_n})||_2$
    $p = Z_{\mathcal{F}_n}(\overline{x}_{\mathcal{F}_n} - x^{(n)}_{\mathcal{F}_n})$
    Calculate α_i by (5.8).
    $\alpha = 1$
    ***while** $q(P[x^{(n)} + \alpha p]) > q(x^{(n)})$*
      $\alpha = \max(\alpha_i < \alpha)$
    end while
    $x^{(n+1)} = P[x^{(n)} + \alpha p]$
    Update the sets $\mathcal{F}_n$, $\mathcal{B}_n$.
  end while
  Calculate Lagrange multipliers.
  Release infeasible variables from $\mathcal{B}$.
end while

The requirement that the objective function must decrease during the iterations can easily be implemented into the algorithm proposed by Portugal. This is done in Algorithm 5.5.

5.4.1 Convergence proof of the BLOCK3 method

In the convergence proof for the single pivoting strategy, we relied on that the objective function was reduced between two successive stationary points. This is not achieved in BLOCK1 and BLOCK2, but the BLOCK3 algorithm was designed with this as a criteria. This implies that convergence in the same way can be proved.

THEOREM 5.4 *The BLOCK3 algorithm converges in a finite number of iterations.*

Proof: The inner loop terminates in a finite number of steps, by the same reason as in Lemma 5.1. Each stationary point is a global solution to the equality constrained problem, $\min q(x)$ with $x_i = l_i$ or u_i, $i \in \mathcal{B}$. The objective function

ALGORITHM **5.5 (BLOCK4. Extension of the BLOCK2.)**

Choose a column permutation P and permute A, l and u
Initiate the index sets $\mathcal{F}_0 = \{1 : n\}$ and $\mathcal{B}_0 = 0$
***while** not optimal point*
solve $\min_{x_{\mathcal{F}_n}} ||A_{\mathcal{F}_n} x_{\mathcal{F}_n} - (b - A_{\mathcal{B}_n} x_{\mathcal{B}_n})||_2$
$p = Z_{\mathcal{F}_n}(\overline{x}_{\mathcal{F}_n} - x^{(n)}_{\mathcal{F}_n})$
Calculate α_i by (5.8)
$\alpha = 1$
***while** $q(P[x^{(n)} + \alpha p]) > q(x_n)$*
$\alpha = max(\alpha_i < \alpha)$
end while
$x^{n+1} = P[x^{(n)} + \alpha p]$
Calculate Lagrange multipliers y, v
Update the sets $\mathcal{F}_n$, $\mathcal{B}_n$
end while

value decreases by the construction of the algorithm and there is only a finite number of equality constrained problems. Therefore the algorithm will converge in a finite number of steps to a point satisfying the KKT conditions. □

5.5 Rank deficient problems

Here we will outline a method for finding the minimum norm solution of the BLS problem when A is rank deficient and the null-space to A has low dimension.

When A is rank deficient, $\text{rank}(A) = r < n$, the solution to the BLS problem may have multiple solutions, see Figure 5.4. However, there is always a unique solution of minimum norm. The corresponding minimum norm problem can be stated as follows,

$$\begin{aligned} &\min_w ||x + w||_2, \\ \text{s. t.} \quad &l \leq x + w \leq u, \\ &w \in \mathcal{N}(A), \end{aligned} \tag{5.9}$$

where x is any solution to the BLS problem. To solve this problem a basis of $\mathcal{N}(A)$ is needed. One way to obtain a representation of the basis is the following. Suppose we have the factorization,

$$Q^T A = \begin{pmatrix} R & S \\ 0 & 0 \end{pmatrix},$$

where R has full column rank. To obtain a matrix W whose columns spans $\mathcal{N}(A)$, we can solve $AW = 0$. By using the factorization above we get,

$$Q^T A W = RW_1 + SW_2 = 0,$$

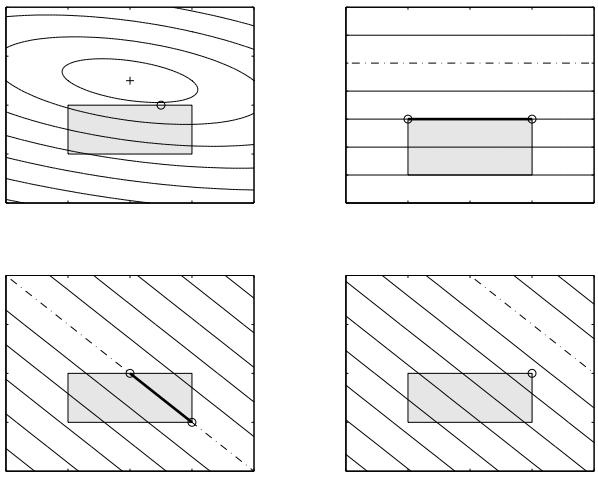

Figure 5.4: Different solutions with a rank deficient A. In the upper left plot A has full rank. The chain line marks the unconstrained optimal solution and if the constrained optimum is not unique the solutions is marked with a thick line.

$W_1 \in \mathbb{R}^{r\times p}$, $W_2 \in \mathbb{R}^{p\times p}$ and $p = n - r$. Since R has full column rank we may choose the matrix W_2 arbitrary. If we let $W_2 = -I_p$ then $W_1 = R^{\perp 1}S$. The matrix $W = (W_1^T \; W_2^T)^T$ is now a representation of the basis for $\mathcal{N}(A)$. W can now be orthogonalized with the modified Gram-Schmidt orthogonalization method to a matrix U. However, U will almost certainly be full. However, since $\mathcal{N}(A)$ is of low dimension, this will not cause a problem. If we can compute the Q factor from the QR factorization, we can use the last p columns in Q instead.

When an orthogonal basis for the null-space of A is known the problem (5.9) can be reformulated. The vector x can be split into two mutually orthogonal parts $x = x_{\mathcal{R}} + x_{\mathcal{N}}$, $x_{\mathcal{R}} \in \mathcal{R}(A^T)$, $x_{\mathcal{N}} \in \mathcal{N}(A)$. Since $||x||_2^2 = ||x_{\mathcal{R}}||_2^2 + ||x_{\mathcal{N}}||_2^2$ we obtain the problem

$$\begin{aligned} &\min_{x_{\mathcal{N}}} ||x_{\mathcal{N}}||_2, \\ \text{s. t.} \quad &l \le x_{\mathcal{N}} + x_{\mathcal{R}} \le u, \\ &x_{\mathcal{N}} \in \mathcal{N}(A). \end{aligned} \tag{5.10}$$

Since $\mathcal{N}(A)$ is spanned by $U \in \mathbb{R}^{n\times p}$ there is a z such that $Uz = x_{\mathcal{N}}$ therefore (5.10) becomes

$$\begin{aligned} &\min_{z} ||z||_2, \\ \text{s. t.} \quad &l \le Uz + x_{\mathcal{R}} \le u, \end{aligned} \tag{5.11}$$

where $z \in I\!R^p$. The part of x in $\mathcal{R}(A)$ must also be known, but it can be computed by $x_{\mathcal{R}} = (I - UU^T)x$.

This is a dense least distance problem with an quadratic objective function of p variables. It can be solved with an active set method for dense QP problems. An algorithm for this problem can be found in Lötstedt [42] based on a QP algorithm by Gill and Murray [29].

5.6 Implementation issues

In this section we will address some of the implementations issues for the active set methods. We will discuss when we should update the factorization and when to refactorize. We will also discuss the termination criteria and the criteria to switch to the single pivoting mode in the block methods that uses this to ensure convergence. The choice of factorization method will also be discussed.

The single pivoting methods are attractive for dense matrix computation. When dealing with dense matrices the number of unknown are usually not as great as with sparse computation. The updating methods for dense matrices are also very efficient. This together with the first sparse QR methods based on a row sequential scheme for computing the QR factorization, led to the use the single pivoting algorithm together with the efficient updating technique developed by Björck [4]. When the multi-frontal methods for computing the QR factorization were developed, more interest for methods that could exploit the efficiency of this new method. We will now present some arguments for and against the choice of block methods instead of single pivoting methods;

- Higher performance can be obtained by using block operations (BLAS 2-3) instead of manipulating single elements. This is especially true when using MATLAB as an implementation language.
- A code using block operations is easier to parallelize.
- Multi-frontal techniques are more efficient for factorization of sparse matrices. However, there are no efficient ways yet, to update the QR factorization using multi-frontal techniques.
- There has to be big differences in concurrent blocks for block schemes to be efficient.

The best thing should be to combine the different approaches, i.e., to use updating techniques when there is a small change in the number of variables and refactorize when large changes occur. In Portugal et al. [62] a heuristic rule for determining if the R factor should be updated or recomputed is given. They use the quantity

$$\lambda = \frac{|\mathcal{L} \cup \mathcal{U} \cup \mathcal{H}|}{|\mathcal{F} \setminus (\mathcal{L} \cup \mathcal{U} \cup \mathcal{H})|},$$

where $|\cdot|$ denotes the size of a set. $\mathcal{H}$ is the index set of Lagrangian multipliers to variables x_i, $i \in \mathcal{F}$ with negative sign. They conclude that if $\lambda > 0.2$ it is

cheaper to recompute R from scratch; otherwise an update should be done. This heuristic value is determined using a sparse row sequential QR factorization, described in [21] and should be higher if the a multi-frontal code is used. We have not tested the heuristic, but believe that in MATLAB, the heuristic value should also be much higher since there is a great performance loss when manipulating sparse matrices not using built-in functions.

To be able to terminate the iteration process in both single pivoting and block methods, we use a stopping criteria based on the Lagrange multipliers. If all variables at their bounds have Lagrange multipliers larger then $> -tol$ and all variables in the free set is feasible, we terminate the iteration process. If is tempting to set tol to zero since the first order optimality conditions state that the Lagrange multipliers should be nonnegative. We have noticed that this tolerance will work for the nondegenerate problems since the Lagrange multipliers corresponding to active constraints are bounded away from zero. However, for the degenerate case, when the active set is not unique and the Lagrange multipliers could be zero, then the algorithms usually do not stop, even if the solution is found. The algorithms will usually bind and release degenerated variables in a cyclic manner.

The choice of tol did not make any differences for the nondegenerate problems. However, we noticed differences in the number of iterations when solving degenerated test problems. We found that by using a tolerance of $n\mathbf{u} \cdot 10^2$ or $n\mathbf{u} \cdot 10^3$, where n is the number of variables, the methods converged fast and gave good accuracy for our test problems.

To decide when the block methods can switch to single pivoting mode we have used the criteria to switch if no reduction of the infeasible variables has occurred in $T = 3$ iterations. This is more strict than in Portugal et al. [62] where they used $T = 10$. We tried different values of this parameter but our choice was best for our problems. In single pivoting mode we have used the quantity (5.6) to choose the next index to free. This was also used in the single pivoting algorithm, `single`.

The test problems presented in section 4 all have full rank and is not ill-conditioned. Therefore we can compute $R_{\mathcal{F}}$ by a Cholesky factorization of $A_{\mathcal{F}}^T A_{\mathcal{F}}$. The least squares problem in each iteration is then solved with the method of normal equations (NE). In each iteration we solve for a descent direction. High accuracy of this direction is not crucial for the convergence of the method, since it is the partition of the variables at the solution we want. However, as the last step we recompute the solution by using the QR factorization of $A_{\mathcal{F}}$. This ensures good accuracy of the solution if the active set is correct. This last step will degrade the performance of the active set method implemented in MATLAB. Since, the Cholesky factorization in MATLAB since the `chol` is very fast, up to 10 times the speed of the QR factorization routine `qr`. This fact does not affect the comparison done in the next section since we have used the recomputation in all algorithms. However, in the final comparison with the interior-point method we have to keep this in mind.

5.7 Numerical experience

In this section we will start with results for the single pivoting algorithm `single` in section 5.3. These results motivate the use of the projected starting point in the single pivoting algorithm. Then we will compare the different block methods introduced in section 5.4

For all the numerical experiments in this section we used MATLAB V5 on an ALPHA STATION 200 4/166, 64 M-byte memory with operating system OSF1 V4.0. The test problems were generated according to Section 4.2. The optimal point was detected when all variables were feasible and the Lagrangian variables were larger then $-tol$ where $tol = n\mathbf{u} \cdot 10^2$. The columns of A were ordered by the minimum degree ordering to reduce fill in R.

5.7.1 Single pivoting algorithms

In Table 5.1 the number of iterations needed for convergence for the single pivoting algorithm are presented. No timings are given since the updating technique described by Björck [4] was not used. For some of the problems we see that the initial guess $P[x^*]$ is very good and only a few iterations are needed to obtain the optimal value. In test problems 10 and 11 the number of iterations needed to converge when all variables were in $\mathcal{B}$ from the beginning is at least 10 times the number needed in table below. Problems of type A are nondegenerated and type B degenerated.

Table 5.1: Number of iterations for the single pivoting algorithm. The test problems of type A and B were generated according to section 4.2

Nr.	1	2	3	4	5	6	7	8	9
A	33	6	5	16	23	285	298	274	259
B	32	26	3	5	9	328	467	260	226

Nr.	10	11	12	13	14	15	16	17
A	11	14	513	548	318	2040	5853	>17263
B	58	2	510	780	252	1843	4131	>17263

5.7.2 Block methods

Here we will compare how the different block methods perform on a set of test problems. Both block and single pivoting steps are computed by a refactorization of R and not with the updating technique presented by Björck [4]. However, for most of the problems this did not influence the computational time since single pivoting occurred only for problem 7 and 13.

In Table 5.2–5.4 the result from the tests are accounted for. In table 5.2 we give the number of factorizations for the different block methods. This number does not include the last QR factorization since that is optional. We can conclude from this table that the overall performance of all the algorithms

are very good. Usually only 3 to 7 factorizations are needed to solve the bounded least squares problem. For the problems 18–26, corresponding to finite element matrices, each method uses almost a constant number of factorizations. This indicates that the methods are not dependent on the size of the problems. For most of the test problems a full step $\alpha = 1$, is taken in each iteration. This means that `BLOCK3` and `BLOCK4` will behave in the same manner as `BLOCK2`. The `BLOCK3` algorithm can in the worst case act like the single pivoting algorithm, but this was not noticed in our test problem suite. We tried to use the QR algorithm to compute R, but we did not notice any difference in the number of factorizations needed to solve the test problems. The accuracy was also exactly the same since we recomputed the solution with the QR method as the last step.

For problem 7 the step length was reduced during several concurrent steps. This interested us so we made a comparison with the same matrix and limits, but with different random solutions. Then we used the `BLOCK2` and `BLOCK3` algorithm to compute the solution, see Table 5.5. The conclusion from this table is that the `BLOCK2` algorithm can have an order of magnitude more factorizations than the `BLOCK3` method. However, this is not as bad as one may think, because most factorizations are from the single pivoting steps. It is therefore very important to use an efficient updating technique for the R factor if the `BLOCK2` algorithm is used. Otherwise we will lose a great deal of efficiency for some problems. The `BLOCK3` algorithm was forced to reduce the step length during several steps to ensure reduction of the objective function. However, this approach did indeed prove more efficient in this case.

In Table 5.3 we show the relative forward accuracy measured in the 2-norm. For the nondegenerate case, type A problems, we obtain the same active set for all algorithms and therefore the accuracy is the same for all test problems. All the solutions were obtained with a satisfactory high accuracy when considering to the condition number of the matrix. For the degenerate problems there is not a unique active set at the solution and therefore we obtain slightly different accuracy for the different algorithms. However, for all the degenerate problems we obtained a solution with an accuracy comparable to the solution of the corresponding nondegenerate problem. We clearly noticed that by refactorizing R with QR as a last step did indeed increase the accuracy of the solution. We strongly recommend this procedure.

Table 5.4 show the computational time measured in seconds for the complete algorithm. We see that for problems 1–17 the `BLOCK2` method was almost always the fastest algorithm. This is due to the simplicity of the algorithm and that it used the same number of factorizations or fewer than the `BLOCK3` and `BLOCK4` algorithms, except for problem 7, type B. For this particular problem `BLOCK2` was faster than `BLOCK3` and `BLOCK4` even if `BLOCK2` used 6 respectively 11 additional factorizations. However, this problem is of small size, so the extensive search for the next descent point became relatively time consuming and made the `BLOCK3` and `BLOCK4` algorithms slow. For the larger finite element problems the `BLOCK3` and `BLOCK4` algorithms were faster than the `BLOCK2` algorithm by using fewer factorizations.

Figures 5.5 and 5.6 show the number of factorizations and the computational

time for all test problems relative the BLOCK1 algorithm.

We conclude that overall the heuristic algorithm BLOCK2 is the most efficient algorithm. However, it should be used with an efficient technique for updating R when columns are added or removed in A. Algorithm BLOCK3 is almost as effective as BLOCK2 for the smaller problems and when the factorization time is more dominant as in test problem 25 and 26 it is more efficient than BLOCK2. We would recommend to use BLOCK2 when an efficient updating technique is implemented otherwise, use the BLOCK3 algorithm.

Table 5.2: Comparison of the different block methods. Number of factorizations.

	Nondegenerate, A				Degenerate, B			
Prob.	B.1	B.2	B.3	B.4	B.1	B.2	B.3	B.4
1	5	4	4	4	6	4	4	4
2	6	4	4	4	5	3	3	3
3	3	3	3	3	3	3	3	3
4	4	3	3	3	3	3	3	3
5	4	4	4	4	3	3	3	3
6	9	5	5	5	9	6	6	6
7	22	10	14	15	30	25	19	14
8	9	6	6	6	10	6	6	6
9	7	5	5	5	6	5	5	5
10	4	3	3	3	4	3	3	3
11	4	3	3	3	3	3	3	3
12	11	7	7	7	11	7	7	7
13	22	14	13	25	15	12	14	12
14	8	5	5	5	11	6	6	6
15	14	8	8	8	17	8	8	8
16	17	9	9	9	25	10	10	10
17	17	9	9	9	18	8	8	8
18	4	4	4	4	4	4	4	4
19	7	5	5	5	6	5	4	4
20	5	6	5	5	5	5	5	5
21	7	6	5	5	8	6	5	5
22	6	6	5	5	7	5	5	5
23	6	6	5	5	7	5	5	5
24	6	6	5	5	15	5	5	5
25	7	6	5	5	6	5	5	5
26	7	6	5	5	7	6	5	5

Table 5.3: Comparison of the different block methods. Relative error in solution. In the nondegenerate case the same solution was obtained.

	Nondegen., A	Degenerate, B			
Prob.	BLOCK1–4	BLOCK1	BLOCK2	BLOCK3	BLOCK4
1	$2.3 \cdot 10^{\perp 16}$	$4.0 \cdot 10^{\perp 16}$	$2.8 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$
2	$1.6 \cdot 10^{\perp 16}$	$2.0 \cdot 10^{\perp 16}$	$2.5 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$
3	$1.8 \cdot 10^{\perp 16}$	$2.8 \cdot 10^{\perp 16}$	$2.2 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$	$2.3 \cdot 10^{\perp 16}$
4	$2.0 \cdot 10^{\perp 16}$	$2.4 \cdot 10^{\perp 16}$	$2.7 \cdot 10^{\perp 16}$	$2.4 \cdot 10^{\perp 16}$	$2.4 \cdot 10^{\perp 16}$
5	$2.6 \cdot 10^{\perp 16}$	$2.1 \cdot 10^{\perp 16}$	$2.6 \cdot 10^{\perp 16}$	$2.8 \cdot 10^{\perp 16}$	$2.8 \cdot 10^{\perp 16}$
6	$1.9 \cdot 10^{\perp 14}$	$1.0 \cdot 10^{\perp 14}$	$1.2 \cdot 10^{\perp 14}$	$1.2 \cdot 10^{\perp 14}$	$1.2 \cdot 10^{\perp 14}$
7	$3.0 \cdot 10^{\perp 10}$	$3.2 \cdot 10^{\perp 10}$	$3.2 \cdot 10^{\perp 10}$	$2.9 \cdot 10^{\perp 10}$	$3.2 \cdot 10^{\perp 10}$
8	$1.1 \cdot 10^{\perp 15}$	$7.3 \cdot 10^{\perp 16}$	$7.7 \cdot 10^{\perp 16}$	$1.0 \cdot 10^{\perp 15}$	$1.0 \cdot 10^{\perp 15}$
9	$1.3 \cdot 10^{\perp 15}$	$1.0 \cdot 10^{\perp 15}$	$1.2 \cdot 10^{\perp 15}$	$1.0 \cdot 10^{\perp 15}$	$1.0 \cdot 10^{\perp 15}$
10	$8.1 \cdot 10^{\perp 16}$	$7.0 \cdot 10^{\perp 16}$	$8.1 \cdot 10^{\perp 16}$	$8.1 \cdot 10^{\perp 16}$	$8.1 \cdot 10^{\perp 16}$
11	$1.0 \cdot 10^{\perp 15}$	$8.5 \cdot 10^{\perp 16}$	$8.8 \cdot 10^{\perp 16}$	$7.2 \cdot 10^{\perp 16}$	$7.2 \cdot 10^{\perp 16}$
12	$1.7 \cdot 10^{\perp 14}$	$3.5 \cdot 10^{\perp 14}$	$1.5 \cdot 10^{\perp 14}$	$1.6 \cdot 10^{\perp 14}$	$1.6 \cdot 10^{\perp 14}$
13	$7.4 \cdot 10^{\perp 11}$	$5.8 \cdot 10^{\perp 11}$	$2.6 \cdot 10^{\perp 11}$	$3.7 \cdot 10^{\perp 11}$	$2.0 \cdot 10^{\perp 11}$
14	$1.8 \cdot 10^{\perp 15}$	$2.0 \cdot 10^{\perp 15}$	$2.0 \cdot 10^{\perp 15}$	$2.0 \cdot 10^{\perp 15}$	$2.0 \cdot 10^{\perp 15}$
15	$5.6 \cdot 10^{\perp 16}$	$8.9 \cdot 10^{\perp 16}$	$1.0 \cdot 10^{\perp 15}$	$6.0 \cdot 10^{\perp 16}$	$6.0 \cdot 10^{\perp 16}$
16	$5.5 \cdot 10^{\perp 08}$	$3.6 \cdot 10^{\perp 08}$	$3.6 \cdot 10^{\perp 08}$	$3.6 \cdot 10^{\perp 08}$	$3.6 \cdot 10^{\perp 08}$
17	$1.8 \cdot 10^{\perp 15}$	$1.7 \cdot 10^{\perp 15}$	$1.9 \cdot 10^{\perp 15}$	$1.7 \cdot 10^{\perp 15}$	$1.7 \cdot 10^{\perp 15}$
18	$5.5 \cdot 10^{\perp 16}$	$5.0 \cdot 10^{\perp 16}$	$7.6 \cdot 10^{\perp 16}$	$6.1 \cdot 10^{\perp 16}$	$6.1 \cdot 10^{\perp 16}$
19	$7.1 \cdot 10^{\perp 16}$	$7.4 \cdot 10^{\perp 16}$	$7.3 \cdot 10^{\perp 16}$	$6.5 \cdot 10^{\perp 16}$	$6.5 \cdot 10^{\perp 16}$
20	$7.9 \cdot 10^{\perp 16}$	$7.4 \cdot 10^{\perp 16}$	$7.0 \cdot 10^{\perp 16}$	$7.6 \cdot 10^{\perp 16}$	$7.6 \cdot 10^{\perp 16}$
21	$8.3 \cdot 10^{\perp 16}$	$8.2 \cdot 10^{\perp 16}$	$7.7 \cdot 10^{\perp 16}$	$8.1 \cdot 10^{\perp 16}$	$8.1 \cdot 10^{\perp 16}$
22	$8.7 \cdot 10^{\perp 16}$	$8.6 \cdot 10^{\perp 16}$	$8.6 \cdot 10^{\perp 16}$	$8.2 \cdot 10^{\perp 16}$	$8.2 \cdot 10^{\perp 16}$
23	$8.8 \cdot 10^{\perp 16}$	$8.5 \cdot 10^{\perp 16}$	$8.9 \cdot 10^{\perp 16}$	$8.8 \cdot 10^{\perp 16}$	$8.8 \cdot 10^{\perp 16}$
24	$8.5 \cdot 10^{\perp 16}$	$9.1 \cdot 10^{\perp 16}$	$8.9 \cdot 10^{\perp 16}$	$8.6 \cdot 10^{\perp 16}$	$8.6 \cdot 10^{\perp 16}$
25	$9.8 \cdot 10^{\perp 16}$	$9.6 \cdot 10^{\perp 16}$	$9.8 \cdot 10^{\perp 16}$	$8.9 \cdot 10^{\perp 16}$	$8.9 \cdot 10^{\perp 16}$
26	$9.6 \cdot 10^{\perp 16}$	$9.3 \cdot 10^{\perp 16}$	$1.0 \cdot 10^{\perp 15}$	$9.7 \cdot 10^{\perp 16}$	$9.7 \cdot 10^{\perp 16}$

Table 5.4: Comparison of the different block methods. Computational time is given in seconds.

	Nondegenerate, A				Degenerate, B			
Prob.	B.1	B.2	B.3	B.4	B.1	B.2	B.3	B.4
1	0.30	0.23	0.30	0.30	0.23	0.18	0.23	0.23
2	0.12	0.10	0.13	0.13	0.12	0.083	0.10	0.12
3	0.10	0.10	0.12	0.13	0.10	0.10	0.13	0.12
4	0.18	0.15	0.18	0.18	0.17	0.18	0.20	0.20
5	0.28	0.28	0.32	0.33	0.25	0.28	0.32	0.32
6	0.63	0.43	0.50	0.50	0.63	0.52	0.58	0.60
7	1.1	0.67	1.0	1.1	1.5	1.4	2.3	2.2
8	0.95	0.80	0.85	0.83	1.0	0.82	0.93	0.92
9	0.93	0.82	0.92	0.9	0.9	0.85	0.90	0.92
10	1.4	1.3	1.3	1.3	1.4	1.2	1.3	1.3
11	1.8	1.7	1.7	1.7	1.8	1.8	1.7	1.8
12	1.5	1.2	1.4	1.3	1.7	1.3	1.4	1.4
13	2.3	1.7	2.9	5.1	1.9	1.8	3.4	2.5
14	6.4	5.7	6.0	6.0	7.7	6.1	6.2	6.2
15	2.8	2.1	2.6	2.7	4.2	2.9	2.9	2.9
16	14	10	11	11	27	16	18	17
17	75	63	74	74	98	85	90	90
18	0.13	0.13	0.18	0.17	0.13	0.15	0.17	0.17
19	0.8	0.68	0.75	0.75	0.75	0.72	0.67	0.67
20	2.0	2.1	2.1	2.1	2.0	2.0	2.2	2.1
21	4.7	4.4	4.2	4.2	5.1	4.7	4.4	4.4
22	8.2	8.2	7.8	7.7	9.2	8.1	8.2	8.0
23	13	13	12	12	14	13	13	12
24	19	19	18	18	33	20	19	19
25	32	30	29	29	33	32	30	30
26	43	41	40	40	46	47	42	42

Table 5.5: Number of factorizations for 10 test problem constructed with the same matrix ILLC1033 (test matrix 7) but with different random solutions x. The problems were degenerated, type B.

Method	Factorizations									
`BLOCK2`	106	33	140	94	114	213	143	22	21	16
`BLOCK3`	19	21	21	15	22	16	23	27	16	15

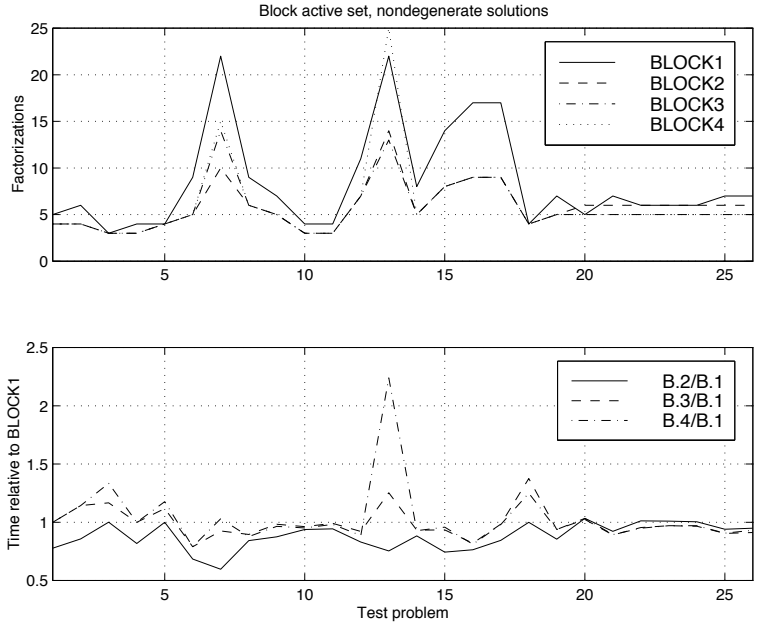

Figure 5.5: Nondegenerate test problems corresponding to Table 5.2, 5.4.

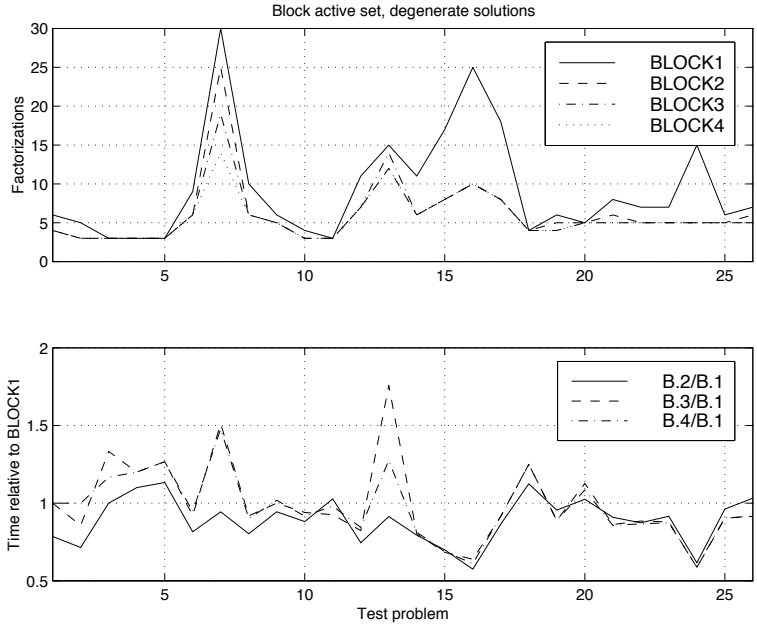

Figure 5.6: Degenerate test problems corresponding to Table 5.2, 5.4.

6 A path following predictor-corrector interior-point method

The interior-point methods got a dramatic attention of the scientists in the middle of 1980s. When Karmakar published his famous paper about a method for solving linear programming problems in polynomial time, it made the first page in New York Times. The area exploded, and there was a revival of the barrier methods from the 60's, since many of the interior-point methods have their origin in barrier function methods. Here we will discuss a certain class of the interior-point methods called *path following*. Our main interest is in a method introduced by Mehrotra. The class of interior-point methods based on potential reduction methods will not be discussed here.

6.1 Barrier function

By adding a penalty function that grows large when we approach a constraint we can transform a constrained optimization problem to an unconstrained problem. A *barrier* penalty function can be constructed by placing a singularity at the constraints. The singularity is regulated with a barrier parameter, $\mu > 0$, and when μ tends to zero the original problem is obtained in the limit. By using a sequence of decreasing μ_i, we solve a sequence of global optimization problems and in the limit we obtain the constrained optimum. The barrier term forces the solution away from the boundaries of the domain, to be *centered* in the domain.

Consider the general convex programming problem,

$$\min_x q(x) \qquad s.\,t. \quad c_i(x) \geq 0,\ i = 1,\ \ldots, m. \tag{6.1}$$

The two most important barrier functions, the *logarithmic barrier* function and the *inverse barrier* function, are expressed in the following way,

$$\begin{aligned} B(x,\mu) &= q(x) - \mu \sum_{i=1}^{m} \ln(c_i(x)), \\ \hat{B}(x,\mu) &= q(x) + \mu \sum_{i=1}^{m} (c_i(x))^{\perp 1}. \end{aligned} \tag{6.2}$$

The logarithmic barrier function was introduced 1955 by Frisch and analyzed in the 1960s. Unfortunately the barrier functions methods suffer from numerical difficulties when approaching the limit point and the methods lost interest during the 1970s. However, the close relationship of the barrier methods with the polynomial time methods as Karmakars, have brought back the interest. The generic barrier algorithm, in simple terms, consists of choosing a strictly feasible point x_1 and $\mu_1 > 0$. The unconstrained optimum is now computed with for example, Newton's method. If the solution do not satisfy the prescribed accuracy we choose a new value μ_2 satisfying $0 \leq \mu_2 < \mu_1$, and the unconstrained optimum recomputed. The convergence of the logarithmic barrier method for

convex programming problems is formulated in the following theorem from M. H. Wright [68].

THEOREM 5 (M. H. WRIGHT [66, PP. 360–361]) *Consider the convex program of minimizing $f(x)$ subject to $c_i(x) \geq 0$, $i = 1, \ldots, m$. Let $\mathcal{F}$ denote the feasible region for this problem, and assume that strict($\mathcal{F}$) is nonempty. Let $\{\mu_k\}$ be a decreasing sequence of positive barrier parameters such that $\lim_{k\to\infty} \mu_k = 0$. Assume that the set $\mathcal{M}$ of constrained local minimizers of the convex program is nonempty and bounded, and let f^* denote the optimal value of f. Then*

(i) the logarithmic barrier function $B(x, \mu_k)$ is convex in strict($\mathcal{F}$);

(ii) $B(x, \mu_k)$ has a finite unconstrained minimizer in strict($\mathcal{F}$) for every $\mu_k > 0$, and the set $\mathcal{M}_k$ of unconstrained minimizers of $B(x, \mu_k)$ in strict($\mathcal{F}$) is convex and compact for every k;

(iii) any unconstrained local minimizer of $B(x, \mu_k)$ in strict($\mathcal{F}$) is also a global unconstrained minimizer of $B(x, \mu_k)$;

(iv) let y_k denote an unconstrained minimizer of $B(x, \mu_k)$ in strict($\mathcal{F}$); then, for all k,

$$f(y_{k+1}) \leq f(y_k) \text{ and } -\sum_{i=1}^{m} \ln c_i(y_k) \leq \sum_{i=1}^{m} \ln c_i(y_{k+1})$$

(v) there exist a compact set $\mathcal{S}$ such that, for all k, every minimizing point y_k of $B(x, \mu_k)$ lies in $\mathcal{S} \cap$strict($\mathcal{F}$);

(vi) any sequence $\{y_k\}$ of unconstrained minimizers of $B(x, \mu_k)$ has at least one convergent subsequence, and every limit point of $\{y_k\}$ is a local constrained minimizer of the convex program;

(vii) let $\{x_k\}$ denote a convergent subsequence of unconstrained minimizers of $B(x, \mu_k)$; then $\lim_{k\to\infty} f(x_k) = f^*$;

(viii) $\lim_{k\to\infty} B_k = f^*$, where B_k denotes $B(x_k, \mu_k)$.

Here strict($\mathcal{F}$) is the set of point in the feasible domain $\mathcal{F}$ where the constraints are satisfied with a strict inequality. If this theorem is applied to our least squares problem with box constraints, then if A has full column rank any sequence of unconstrained minimizes x_k will converge to the unique optimum of the convex programming problem. This is clear since all subsequences has the same convergence point.

An important result from duality theory is the following bound on the optimal solution. If x_k is the unconstrained optimum to $B(x, \mu_k)$, then $0 \leq q(x_k) - q(x^*) \leq m\mu_k$, where m is the number of constraints. This result is independent of the objective function of the convex program and is used later in the interior-point method presented below. The rate of convergence of the

barrier method is clearly related to the choice of the barrier parameter μ_k. How should we choose the next μ_{k+1} and with what accuracy do we need to solve each unconstrained problemΓ Several heuristic methods for choosing μ_{k+1} will be tested later in section 6.2.2.

The optimal point for a fixed value of $\mu = \mu_k$ in $B(x, \mu)$ is given by the conditions that the gradient of B vanishes,

$$\begin{aligned}\nabla B(x, \mu_k) &= g(x) - \sum_{i=1}^{m} \frac{\mu}{c_i(\mu)} d_i \\ &= g(x) - \mu D^T C^{\perp 1}(\mu) e,\end{aligned}$$

where $g(x)$ is the gradient of $q(x)$, D is the Jacobian matrix of $c(x)$, $C = \text{diag}(c_i)$ and e is a vector with all ones. This implies that the gradient of $q(x)$ can be written as a linear combination of the gradients of the constraints. The KKT condition (1.5) says that the gradient at the optimum of $q(x)$ is a linear combination of the active constraints. By this analogy we can suspect that the coefficient $\mu/c_i(\mu)$ is direct analogous to the Lagrangian multiplier λ_i. It can be shown that if the sufficient optimality conditions in Theorem 1.6 hold and the gradients of the active constraints are linearly independent, the quantity $\mu/c_i(\mu)$ converges to the Lagrangian multipliers. Under these conditions there exist a differentiable path $\Gamma = x(\mu)$ in the neighborhood of $\mu = 0$. In our problem of interest it can be shown that the path exists for all $\mu > 0$ when A has full column rank. This path is called the *barrier trajectory* or sometimes, in linear programming, the *central path*. In Figure 6.1 an example of this path is shown together with the level set of the quadratic objective function. The constrained optimum is located at (2, 0). In the interior-point methods called path-following methods, the properties of this trajectory is used and the method follows the path to the solution. Since we do not want to solve an unconstrained problem in each iteration, it suffices if the current iteration point is in a neighborhood of the path.

6.2 Primal-Dual interior-point methods

The name Primal-Dual methods refers to methods generating points feasible for both the primal and dual problem. The first to propose a primal-dual method for the linear programming problem was Megiddo 1986. This was later implemented by Kojima, Mizuno and Yoshise 1988. Monteiro and Adler [54, 53] used these ideas and developed a faster algorithm for the linear and quadratic programming problem. This was achieved by using the logarithmic barrier function together with Newton's method. The barrier parameter μ was determined in each iteration as the decreasing sequence $\mu_{k+1} = (1 - \delta/\sqrt{n})\mu_k$, where $\delta \in]0, \sqrt{n}[$. They proved convergence in $\mathcal{O}(n^3 L)$ operations for the quadratic case, where L is a measure of the problem size.

It is not clear how to choose a good way to reduce μ in each step. Ideally we want to reduce μ to zero as fast as possible. This may imply that choosing the next μ_{k+1} dynamically rather than a having a fixed ratio could be more efficient.

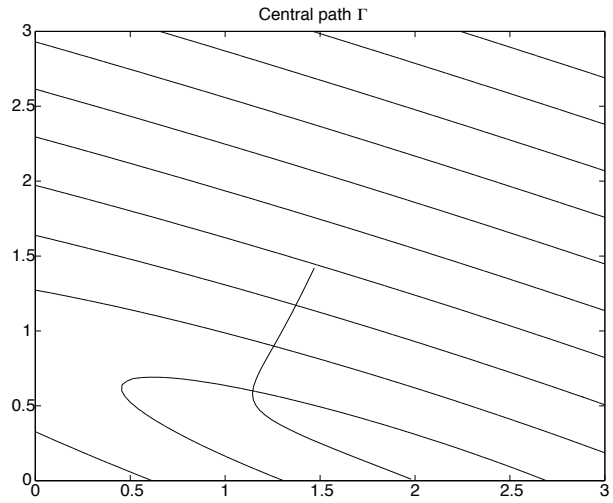

Figure 6.1: The level set of the quadratic objective function together with the central path projected onto the x space. $0 \leq x_{1,2} \leq 3$

Later we will describe a method by Mehrotra that uses the information from the predictor step and the path to determine how well the method is advancing and chooses next μ accordingly. A convergence proof for a primal-dual *short step* methods that uses the fixed factor above can be found in [55, 69]. For a monograph covering primal-dual interior-point methods for the linear programming problem and to a certain extent also quadratic and convex programs, see, S. Wright [69].

Now we will consider problem of BLS. We reformulate this problem as the equivalent quadratic program,

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & q(x) = \tfrac{1}{2} x^T A^T A x - x^T A^T b, \\ \text{subject to} \quad & 0 \leq x_i - l_i, \ i = 1, .., \ l \leq n, \\ & 0 \leq u_i - x_i, \ i = 1, ..., \ k \leq n. \end{aligned} \tag{6.3}$$

The dual problem is given by,

$$\begin{aligned} \underset{(x,v,y)\in\hat{\Omega}}{\text{maximize}} \quad & \tfrac{1}{2} x^T A^T A x - x^T A^T b + v^T (l - x) + y^T (x - u), \\ \hat{\Omega} = \left\{ (x, v, y) \in I\!R^{3\times n} : A^T (Ax - b) + y - v = 0, \ v, y \geq 0, \ x \geq l \right\}. \end{aligned} \tag{6.4}$$

The set $\hat{\Omega}$ defines points (x, v, y) feasible for both the primal and dual problem.

The *complementarity gap* is then given by the difference between primal (6.3) and the the dual (6.4) problem,

$$g(x, v, y) = v^T (x - l) + y^T (u - x). \tag{6.5}$$

The optimality conditions for the problem above is given by the KKT conditions (1.5). Introduce slack variables

$$s = u - x, \quad \text{and} \quad t = x - l. \tag{6.6}$$

If no lower (upper) limit exists the corresponding components in v (y) are set to zero, s and t likewise. The KKT conditions will be the following,

$$\begin{aligned} A^T(Ax-b) - v + y &= 0, \\ x_i - u_i + s_i &= 0, \ i = 1, ..., \ l \leq n, \\ l_i - x_i + t_i &= 0, \ i = 1, ..., \ k \leq n, \\ t_i v_i &= 0, \quad \forall i, \\ s_i y_i &= 0, \quad \forall i, \\ (s, t, v, y) &\geq 0. \end{aligned} \tag{6.7}$$

To simplify the notations later on we introduce the variable

$$w = (s, \ t, \ v, \ y, \ x). \tag{6.8}$$

Observe that this system of equations would be almost identical, if we had a positive definite quadratic programming problem subject to simple bounds in the variables. We will later point out what changes is needed for solving such problems instead of solving the least squares problem.

If we would attack this problem straight on, we could use the Newton method on the nonlinear system defined by (6.7), due to its quadratic convergence rate. When the search direction is obtained we restrict the step length to ensure that the next iterate is in the interior of $\hat{\Omega}$. This is to avoid spurious solutions to (6.7) that can appear if we move outside the domain. However, to apply the Newton's method direct at (6.7) turns out to be impractical if the optimum is not in the interior of the domain. The Newton iterations tend to move very quickly outside the domain and the step restriction will make convergence slow, if it will converge at all.

The primal-dual interior-point method modifies the basic Newton method in two important ways. First the search direction generated by the Newton method is biased toward the interior of the feasible domain. The second important objective is to keep the iterates away from the boundaries of the orthant. The search directions computed near a boundary tend to be distorted.

These two changes makes the method able to take longer steps towards the solution of (6.7). We use the logarithmic barrier function as the means to make the changes. The logarithmic barrier function with the nonnegative barrier parameter μ to (6.3) becomes,

$$\min_x B(x, \mu) = \frac{1}{2} x^T A^T A x - x^T A^T b - \mu \left(\sum_{i=1}^{l} \ln(t_i) + \sum_{i=1}^{k} \ln(s_i) \right), \tag{6.9}$$

where t_i and s_i are slack variables (6.6). $B(x, \mu)$ is a convex function and the domain is convex so it has a unique global optimum for each μ. This optimum

is can be obtain by differentiating (6.9),

$$\nabla_x B(x,\mu) = A^T(Ax-b) - \mu(T^\dagger e - S^\dagger e) = 0.$$

Here T is the diagonal matrix with t_i on the diagonal and zeros when no lower bound exist. $T^\dagger$ is the pseudo-inverse, $t_i^{\perp 1}$ if $t_i \neq 0$ and 0 if $t_i = 0$ on the diagonal, $S^\dagger$ likewise. Let

$$\mu T^\dagger e = Ve, \quad \text{and} \quad \mu S^\dagger e = Ye,$$

then we obtain the following nonlinear system,

$$f_\mu(w) = \begin{pmatrix} A^T(Ax-b) - v + y \\ x - u + s \\ l - x + t \\ TVe - \mu e \\ SYe - \mu e \end{pmatrix} = 0. \tag{6.10}$$

This system of equations is similar to the (6.7) except for the terms μe in the complementarity conditions. Observe that v and y are not the Lagrange multiplier as in (6.7). However, in the limit as $\mu \to 0^+$ v and y will converge to the Lagrange multipliers in (6.7). If A has full column rank, it can be shown with the implicit function theorem that the equation (6.10) defines a continuous differentiable path depending on μ, $\Gamma(\mu) = w_\mu$ where w_μ is the solution to (6.10) for different fixed μ. When $\mu \to 0$ the optimal solution is to the original problem (6.3) is obtained, since by Theorem 5 all sequences $w_\mu \to w^*$.

If Newton's method is applied to (6.10) for a fixed μ, the last two equations will move the search direction away from the boundary of the feasible domain. By introducing a *centering parameter* σ, $\sigma \in [0,\ 1]$, and replacing μ with $\sigma\mu$ in the algorithm, we can control if the search direction will try to move to the solution of (6.7), with $\sigma = 0$ and reduce μ, or move the next iterate toward the solution of the unconstrained problem, the central path, with $\sigma = 1$. When $\sigma = 0$ we call the step an *affine step* and with $\sigma = 1$ a *centering step* is obtained. By using a σ between zero and one we make a combination of moving towards the path and trying to reduce μ. We call μ the *duality measure* and define it as,

$$\mu = \frac{v^T t + y^T s}{2n} = \frac{g(w)}{2n}. \tag{6.11}$$

If we have a point close to the path, this will give an estimations of a μ that yields a point on the path close to the our current point. For a point strictly on the path $\Gamma(\mu)$ this relation can be obtained from the last two equations in (6.10). The short step methods mentioned above use a conservative centering parameter close to one. These methods can usually take a unit step in the Newton direction but make a slow progress towards the solution.

The Newton step for (6.10) is obtained by differentiating f. We get the

following Jacobian,

$$f'(w) = \frac{\partial f(w)}{\partial w} = \begin{pmatrix} 0 & 0 & -I & I & A^TA \\ I & 0 & 0 & 0 & I \\ 0 & I & 0 & 0 & -I \\ 0 & V & T & 0 & 0 \\ Y & 0 & 0 & S & 0 \end{pmatrix}. \tag{6.12}$$

The corrections in the Newton step are then obtained by solving the following system,

$$\begin{aligned} -dv + dy + A^TAdx &= A^T(b - Ax) + v - y, \\ ds + dx &= u - x - s, \\ dt - dx &= -l - t + x, \\ Tdv + Vdt &= -TVe + \sigma\mu e, \\ Yds + Sdy &= -YSe + \sigma\mu e. \end{aligned} \tag{6.13}$$

In a predictor-corrector setting the predictor step tries to reduce the duality measure by setting $\sigma = 0$. The corrector step, $\sigma = 1$, will then move the iteration back towards the path Γ to obtain a good starting point for the next iteration. The barrier parameter μ is estimated after the affine step. Each step will require the solution of the linear equation system above. This means an evaluation and factorization in each step. However, we could reuse the factorization in more than one step and save computational effort.

6.2.1 Mehrotra's Predictor-Corrector methods

The most commonly used interior-point codes since 1990 are based on Mehrotra's predictor-corrector algorithm [51]. Motivated by the use of higher order approximations of the central path and ideas of Lustig, Marsten and Shanno [44], Mehrotra describes a method that alternates between taking a predictor (*affine*) step and a corrector (*centering*) step. In addition to this he uses a clever way to estimate the progress of the Newton direction in the predictor step and uses this information in the correction step. The factorization from the affine step is reused in the corrector step, but also a correction for the nonlinearity is added to the correction step. In [52] Mehrotra used a second-order Taylor expansion of the central path to motivate the algorithm. Mehrotra's algorithm is further analyzed and discussed by Lustig et al. [45]. A comparison of the the different linear programming problems from Netlib[6], is also given there. They report that Mehrotra's algorithm outperformed the primal-dual method (OB1 [44]) on iteration count in 85 out of 86 problems and that the execution time was smaller for 71 of the 86 problems.

Carpenter et al. [8] discuss the use of multiple corrections in each iteration and show the equivalence to the composite Newton method and generalize

[6]URL: http://www.netlib.org

Mehrotra's algorithm for quadratic programming problems. Tapia et al. [65] also discusses the equivalence to a damped level-1 composite Newton method.

The algorithm proposed by Mehrotra uses the predictor step to adaptively determine the parameter σ with a heuristic rule. If a large reduction of the complementarity gap, $g(w)$ was made, the current point is well centered and the corrector step can be used to reduce $g(w)$ further. Otherwise the correction step uses a σ close to one to center the iteration to the path. Mehrotra's method uses the same factorization to compute the predictor and the corrector step but makes the following correction for nonlinearity in the corrector step. Consider equation (6.10) with the current iterate and the correction so that $w+dw$ solves (6.10). We then have,

$$f(w+dw) = \begin{pmatrix} A^T(A(x+dx)-b)-(v+dv)+(y+dy) \\ (x+dx)-u+(s+ds) \\ l-(x+dx)+(t+dt) \\ (T+dT)(V+dV)e-\mu e \\ (S+dS)(Y+dY)e-\mu e \end{pmatrix} = 0. \tag{6.14}$$

This equation system can easily be rewritten as

$$\begin{aligned} -dv+dy+A^TAdx &= A^T(b-Ax)+v-y, \\ ds+dx &= u-x-s, \\ dt-dx &= -l-t+x, \\ Tdv+Vdt &= -TVe-dTdVe+\sigma\mu e, \\ Yds+Sdy &= -YSe-dSdYe+\sigma\mu e. \end{aligned} \tag{6.15}$$

This is very close to the linearized equation system given by (6.13). The only difference is the nonlinear terms in the last two equations. Mehrotra used the corrections computed from the affine system of equations to approximate these two terms in the corrector step. This will use second order information of the path and also reduce work compared to the usual predictor-corrector methods, which computes a new factorization in each step.

Due to the fact that we cannot stay on the path, we will solve a slightly different equation system in each iteration. The variables x, s, t, v and y will then have the same importance. In other words, the variable s will not be computed in each iteration to satisfy the constraint $u-x+s=0$, but will have the same status as x and be iterated to convergence as all the others. It has been reported in [45] that better stability for problems with tight bounds is obtained if all variables are given the same status. The affine Newton step, $\sigma = 0$ for (6.10) is,

$$\begin{aligned} -dv+dy+A^TAdx &= A^T(b-Ax)+v-y, \\ ds+dx &= u-s-x, \\ dt-dx &= -l-t+x, \\ Tdv+Vdt &= -TVe_1, \\ Yds+Sdy &= -YSe_2, \end{aligned} \tag{6.16}$$

and the full corrector step with the correction of Mehrotra is,

$$\begin{aligned} -d\tilde{v} + d\tilde{y} + A^T A d\tilde{x} &= A^T(b - Ax) + v - y, \\ d\tilde{s} + d\tilde{x} &= u - s - x, \\ d\tilde{t} - d\tilde{x} &= -l - t + x, \\ Td\tilde{v} + Vd\tilde{t} &= -TVe_1 - dTdVe_1 + \sigma\mu e_1, \\ Yd\tilde{s} + Sd\tilde{y} &= -YSe_2 - dYdSe_2 + \sigma\mu e_2. \end{aligned} \tag{6.17}$$

The corrector step is called full because the affine correction is included in the new correction obtained from the system of equations above. We could compute the corrector correction separately and add both corrections to the initial w. However, by solving the corrections by (6.17), it will be easier to implement multiple corrections later. Here e_1 and e_2 are vectors with ones where lower/upper bounds are present and zeros otherwise. In the outline of Mehrotra's predictor-corrector method shown below, τ is slightly smaller than one (usually 0.95 or 0.9995) to keep the next iterate safely away from the boundary.

ALGORITHM **6.1 (Outline of Mehrotra's method)**

do *until convergence.*
  Factorize system of equations $f'(w)$.
  Solve equation (6.16) for dw.
  Estimate $\sigma\mu$ *from* w_n *and* dw.
  Solve equation (6.17) for $d\tilde{w}$.
  Compute step length α.
  $w_{n+1} = w_n + \tau\alpha d\tilde{w}$.
end do

In the algorithm for linear programming problems, Mehrotra used different step length for the primal and dual variables. However, since x appears in the dual constraint in the quadratic programming case, the same step length is taken here to avoid increasing infeasibility. In Figure 6.2 the level set of a quadratic objective function together with the central path and the predictor-corrector iterations, in the x-space, from Mehrotra's algorithm are shown.

6.2.2 Estimating the centering parameter

One of the new features in Mehrotra's algorithm was to estimate the centering needed in each step. The method used in the first paper by Mehrotra [51] was the following heuristic value for linear programming problems,

$$\sigma = \left(\frac{(x - \alpha_x dx)^T (s - \alpha_s ds)}{x^T s} \right)^{\nu}, \tag{6.18}$$

with $\nu = 3$ and dx, ds are the affine corrections from the Newton step corresponding to (6.16) for the linear programming problem. The use of this heuristic

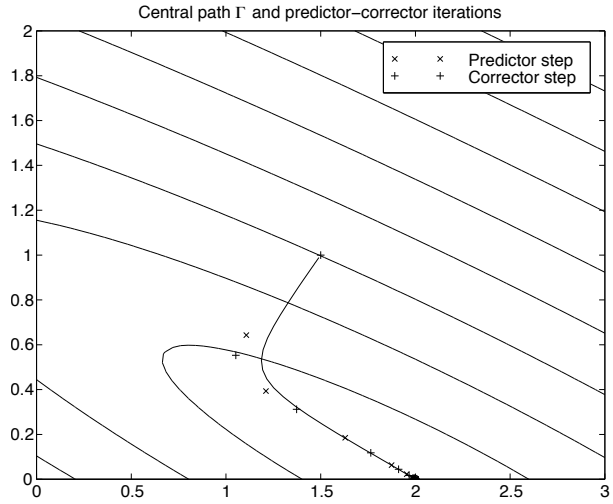

Figure 6.2: The level set of the quadratic objective function together with the central path and predictor-corrector factorizations from Mehrotra's method projected into the x space.

value can be motivated by formulating (6.18) in the following way. Consider,

$$\frac{(x-\alpha_x dx)^T(s-\alpha_s ds)}{x^T s} = \frac{\mu_{\text{aff}}}{\mu}, \tag{6.19}$$

where μ_{aff} is the duality measure from the affine step. If μ_{aff} is much smaller than μ we have obtained a good search direction with the affine direction and in the corrector step we continue to try to decrease μ. On the other hand if μ_{aff} has not decreased significant the affine direction was not good, and we use the corrector step to move closer to the path. Hopefully we then have a better starting point for the next iteration. Later Mehrotra [52] tried the values $\nu = 2,\ 3,\ 4$, but concluded that $\nu = 3$ was the best choice. However, little difference for ν between two and four was reported.

Lustig et al. [45] report that by choosing μ by (6.18) there can be numerical difficulties as the optimum is approached. Instead they used (6.18) with $\nu = 2$ when the complementarity gap $g(w)$ was greater then one, and the following heuristic value when $g(w) < 1$,

$$\sigma\mu = \frac{x^T z + s^T w}{\Phi(n)}; \quad \Phi(n) = \begin{cases} n^2 & \text{if } n \leq 5000, \\ n^{\frac{3}{2}} & \text{if } n > 5000. \end{cases} \tag{6.20}$$

We did not detect any of these difficulties by using (6.19) instead of using (6.20).

In Table 6.1 we show results for a subset of our test problems using the different values of ν together with two other alternatives. The first alternative,

Table 6.1: Number of factorizations from Mehrotra's predictor-corrector algorithm with different heuristic ν in equation (6.18). The alternatives are defined by (6.11) and (6.20).

	Nondegenerate				
Problem	$\nu = 2$	$\nu = 3$	$\nu = 4$	alt. 1	alt. 2
2	10	9	10	10	15
7	11	11	11	14	14
16	13	13	14	14	18
19	11	11	11	12	14
26	11	11	11	12	16

	Degenerate				
Problem	$\nu = 2$	$\nu = 3$	$\nu = 4$	alt. 1	alt. 2
2	24	24	24	24	25
7	25	25	25	25	28
16	30	29	29	33	37
19	25	25	25	25	27
26	25	25	25	25	27

alt 1., is the heuristic value (6.20), proposed by Lustig et al. In the second alternative, alt. 2, we used the duality measure given by (6.11). This is the same as using $\sigma = 1$ in the corrector step. In most cases we get the fewest number of factorizations when we use $\nu = 3$. We will use $\nu = 3$ in all the comparisons from here on. No significant difference in the accuracy of the solution could be found using the different estimates of μ.

6.2.3 Solving for the Newton direction

The main part of the work in the algorithm is in solving for the Newton direction. If A is ill-conditioned we would like to avoid forming A^TA because of the squaring of the condition number. We can avoid this by reducing the system to a more convenient form. The equation systems (6.16)–(6.17) is sparse if A is sparse and this has to be accounted for when making the reformulation of the system. Consider the system from the affine Newton step (6.16),

$$\begin{aligned} -dv + dy + A^T A dx &= A^T(b - Ax) + v - y, && (6.21)\\ ds + dx &= u - s - x, && (6.22)\\ dt - dx &= -l - t + x, && (6.23)\\ Tdv + Vdt &= -TVe, && (6.24)\\ Yds + Sdy &= -YSe. && (6.25) \end{aligned}$$

Multiplying (6.22) with $-Y$ from the left we get

$$-Yds - Ydx = Ys + Y(x-u).$$

Adding this to equation (6.25) and multiplying all equation from the left with $-S^{\perp 1}$ we obtain

$$-dy + S^{\perp 1}Ydx = S^{\perp 1}Y(x-u). \tag{6.26}$$

Make a similar transformation of equation (6.23) but multiply with $-V$ from the left, add (6.24) and multiply from the left with $T^{\perp 1}$. We will get the following,

$$dv + T^{\perp 1}Vdx = T^{\perp 1}V(l-x). \tag{6.27}$$

Now add together (6.26), (6.27) and (6.21) and we get,

$$(A^TA + D)dx = A^T(b-Ax) + v - y + S^{\perp 1}Y(u-x) + T^{\perp 1}V(l-x), \tag{6.28}$$

where $D = S^{\perp 1}Y + T^{\perp 1}V$. This can be interpreted as the normal equation to the following least squares problem,

$$\min_{dx} \left\| \begin{pmatrix} A \\ D^{1/2} \end{pmatrix} dx - \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \right\|_2, \tag{6.29}$$

where

$$b_1 = b - Ax, \quad b_2 = D^{\perp 1/2}\big(v - y + S^{\perp 1}Y(u-x) + T^{\perp 1}V(l-x)\big).$$

If $s = u - l$ and $t = x - l$ then b_2 would be equal to zero, but we let s and t iterate to convergence as pointed out before. The remaining corrections are obtained by,

$$\begin{aligned} ds &= -dx + u - s - x, \\ dt &= dx - l - t + x, \\ dy &= -y - S^{\perp 1}Yds, \\ dv &= -v - T^{\perp 1}Vdt. \end{aligned}$$

In the corrector step the only difference is a new term in b_2. The new right hand side becomes,

$$\tilde{b_2} = b_2 + D^{\perp 1/2}\big(S^{\perp 1}(dYds - \mu e) - T^{\perp 1}(dTdv - \mu e)\big),$$

and the corrections in the corrector step are obtained from,

$$\begin{aligned} d\tilde{s} &= -d\tilde{x} + u - s - x, \\ d\tilde{t} &= d\tilde{x} - l - t + x, \\ d\tilde{y} &= -y - S^{\perp 1}\big(Yd\tilde{s} + dYds - \mu e\big), \\ d\tilde{v} &= -v - T^{\perp 1}\big(Vd\tilde{t} + dTdv - \mu e\big). \end{aligned}$$

The only change in solving for the Newton directions above for a positive definite quadratic programming problem, $\min x^TQx - x^Tc$, subject to simple constrains, would be to change A^TA to Q and A^Tb to c in (6.28) and then we can use the Cholesky factorization to solve the system of linear equations.

6.2.4 Multiple corrections

Carpenter et al. [8] suggested to use each factorization further by performing multiple corrections in each step. Each correction would only cost two triangular solves and if the total number of factorizations would decrease there could be a computational gain in the scheme. They tried the following ideas for selecting the number of corrections for the quadratic programming problem.

- 1-correction. The method suggested by Mehrotra.
- 3-correction. One to three corrections in each step.
- 99-correction. One to 99 corrections in each step
- 0.5 heuristic. One to three corrections. A new correction is tested only if the steplength was greater than 0.5.

For the linear programming problem they also tried several other methods for example 10-correction, another variant of the 99 correction scheme, estimation of μ dynamically in each correction and a heuristic method for choosing if a new correction is needed. We will use these on the quadratic problem and add a scheme depending on the closeness to the path. The generic algorithm for multiple correction scheme with a fixt maximum number of iterations scheme will be,

---

ALGORITHM **6.2 (Outline of multiple correction method)**

do *until convergence.*
  Compute factorization.
  Solve for dw_0: $f_0'(w_n)\delta w_0 = -f_0(w_n)$
  Estimate μ *from* w *and* δw.
  i=0
  while *reduction of* g ***and*** $i < n$.
    i=i+1
    Solve equation (6.17) for $d\tilde{w}_i$.
    Compute step length α_i.
    $w_{n,i} = w_n + \tau\alpha_i dw_i$
    Compute gap, $g(w_{n,i})$.
  end for
  Compute step length α.
  $w_{(n+1)} = w_n + \tau\alpha dw_i$
end do

In the n-correction scheme the new correction was only used if the complementarity gap was reduced. If the validity of the new correction could be determined in advance, much could be saved. In other path following methods not based on Mehrotra's algorithm different neighborhoods of the paths are used to keep the iterations close to the path. The two most popular domains used are,

$$\begin{aligned} \mathcal{N}_2(\theta) &= \{(s,y,t,v) \in \Omega \mid ||SY - \mu e||_2 \leq \theta\mu \text{ and } ||TV - \mu e||_2 \leq \theta\mu\}, \\ \mathcal{N}_{\perp\infty}(\theta) &= \{(s,y,t,v) \in \Omega \mid s_i y_i \geq \theta\mu \text{ and } t_i v_i \geq \theta\mu\}. \end{aligned}$$

The domain $\mathcal{N}_2$ is quite restrictive and is used by short step methods. The $\mathcal{N}_{\perp\infty}$ on the other hand uses only a one-sided bound that pushes the iteration point away from the boundary. It is easy to add the bound $\mathcal{N}_{\perp\infty}$ as a constraint in Mehrotra's algorithm with multiple corrections. One way to do this is to correct until the new point belongs in $\mathcal{N}_{\perp\infty}$. We use $\theta = 0.001$ as suggested by Wright [69].

For all the methods above the $\sigma\mu$ was calculated before the multiple corrections was carried out. In the last method $\sigma\mu$ is recalculated after each calculation and a maximum of three corrections was carried out. This scheme is called dynamic mu-3.

We summarize the different ways to make the multiple corrections below and the results from the test problems can be found in Table 6.2.

- 1-correction. Maximum of 1 correction.
- 2-correction. Maximum of 2 correction.
- 3-correction. Maximum of 3 correction.
- 5-correction. Maximum of 5 correction.
- 10-correction. Maximum of 10 correction.
- 99-correction. Maximum of 99 correction.
- 10-correction, $\mathcal{N}_{\perp\infty}$. Correct until $w^{n+1} \in \mathcal{N}_{\perp\infty}(10^{\perp 3})$ or max 10 corrections.
- dynamic mu-3. Estimate μ dynamically and max 3 corrections.

The relative error for all the different schemes are comparable; very small variations occured. In the degenerate case methods with many corrections often yield a slightly smaller relative error than the single correction algorithm. The computation was done on a SPARC Ultra and the computation times here can not be compared with the active set method in Section 5. In most of the test problems the 1-correction method runs faster then multiple, but for the large problems there can be a gain in multiple corrections due to the expensive factorization cost. For the degenerate case the number of factorizations can be cut by half if a maximum of 99 corrections are used. The limit of 99 corrections was set to reflect an unbounded number of iterations but for all degenerate problems this bound was reached for some iteration.

The method of using corrections until the iteration point was inside the $\mathcal{N}_{\perp\infty}$ did not yield a better result than the initial method. This may depend on the fact that is was only the first iterations that in some cases were outside $\mathcal{N}_{\perp\infty}$. All iterations in the end belonged to the domain.

The conclusion is to use only one correction for small and nondegenerate problems and up to five correnctions for large degenerate problems with a matrix that is expensive to factorize. If the QR factorization is used instead we noted that two to five corrections are preferable to use. This can be explained with the fact that the implementation of sparse QR factorization in MATLAB is undeservedly slow compared with the sparse Cholesky factorization.

Table 6.2: Number of iterations for type A and type B test problems defined in section 4.2.

	Factorizations, type A problems							
Problem	1	2	3	5	10	99	$\mathcal{N}_{\perp\infty}$	dyn μ
2	10	8	8	7	6	6	9	8
7	11	10	9	8	8	8	10	10
16	13	11	10	9	9	9	10	11
19	11	10	9	8	8	8	10	10
26	11	10	9	8	7	6	11	10

	Factorizations, type B problems							
Problem	1	2	3	5	10	99	$\mathcal{N}_{\perp\infty}$	dyn μ
2	24	20	17	15	12	8	22	18
7	25	21	19	17	15	13	24	22
16	30	26	24	23	21	21	27	25
19	25	21	19	16	13	10	23	19
26	25	21	19	16	13	10	24	19

Table 6.3: Computational time for type A and type B test problems defined in section 4.2.

	Time in seconds, type A problems							
Problem	1	2	3	5	10	99	$\mathcal{N}_{\perp\infty}$	dyn μ
2	0.21	0.22	0.25	0.28	0.33	0.45	0.22	0.26
7	0.66	0.75	0.78	0.87	0.96	0.96	0.67	0.82
16	16.33	16.25	16.47	17.41	19.85	19.86	17.83	17.69
19	0.95	1.08	1.11	1.19	1.40	1.59	1.27	1.23
26	46.76	48.57	47.08	45.37	48.93	53.66	47.64	52.44

	Time in seconds, type B problems							
Problem	1	2	3	5	10	99	$\mathcal{N}_{\perp\infty}$	dyn μ
2	0.53	0.57	0.58	0.71	0.97	2.36	0.61	0.64
7	1.43	1.53	1.61	1.84	2.45	4.36	1.53	1.75
16	36.9	38.22	39.39	42.77	50.3	77.31	40.71	40.24
19	2.07	2.19	2.33	2.57	3.36	7.29	2.18	2.31
26	112.0	104.2	103.5	101.7	112.6	204.4	112.1	102.3

6.2.5 Stopping criteria and implementation issues

In the active set methods we could verify that the obtained active set was the optimal by inspecting the Lagrange multipliers. After the active set was determined the solution could be recomputed with a QR factorization to obtain an accurate solution. The interior-point method will only produce points in the interior of the domain and the active bounds will never be exactly satisfied. Therefore a good way is needed to stop the iteration when a sufficiently accurate solution has been found. One quantity for measuring the convergence is the duality gap, $g(w)$, but this only gives information about the closeness of the objective function to the optimal value, not the accuracy in x. The stopping criteria we have used here are based on the duality gap and the primal and dual feasibility. When the following four conditions are satisfied we stop the iteration,

$$\begin{aligned} \frac{t^T v + s^T y}{2n} &< \varepsilon_1, \\ ||x - u + s||_2 &< \varepsilon_2, \\ ||l - x + t||_2 &< \varepsilon_2, \\ ||A^T(Ax - b) + y - v||_2 &< \varepsilon_2. \end{aligned}$$

In Figures 6.3–6.4 the relation between the number of iterations and the relative forward error in 2-norm for different values of ε_1 are plotted for a selected number of test problems. Note that the matrix number 7 is more ill-conditioned than the others. The value of ε_2 is not as important as ε_1, in general it is the first stopping criteria that is the restriction. We see that for the degenerate case we have much slower convergence rate than in the nondegenerate case. For the nondegenerate case the error is improved in a quadratic fashion. This behavior can be expected since Newton's method on unconstrained problems converges at least quadratically for simple roots. From Figure 6.3 we see that for the nondegenerate case a good value of ε_1 is around $10^{\perp 18}$. Then high accuracy has been obtained for all problems. However, for the nondegenerate case we get a acceptable error first when ε_1 is chosen as $10^{\perp 25}$, see Figure 6.4. These values of ε_1 did yield an equally accurate result for problems with other magnitudes of the Lagrange multipliers than the range specified in Section 4.2.

When the algorithm converges the elements in $D^{1/2}$ corresponding to variables bounded at the solution will become very large. It is easy to show that the condition number of $\left(A^T\ D^{1/2}\right)^T$ will tend to infinity if any variables are at their bounds in the solution. Will the computed Newton direction then become inaccurateΓ This question was the big problem for the barrier methods during the 60's.

We now make an informal analysis of what happens when the barrier parameter goes to zero. For simplicity we consider problems with only lower bounds. The reasoning, however, can be applied to both upper and lower bounds. Assume that the problem is nondegenerate and that the current iterate is close the

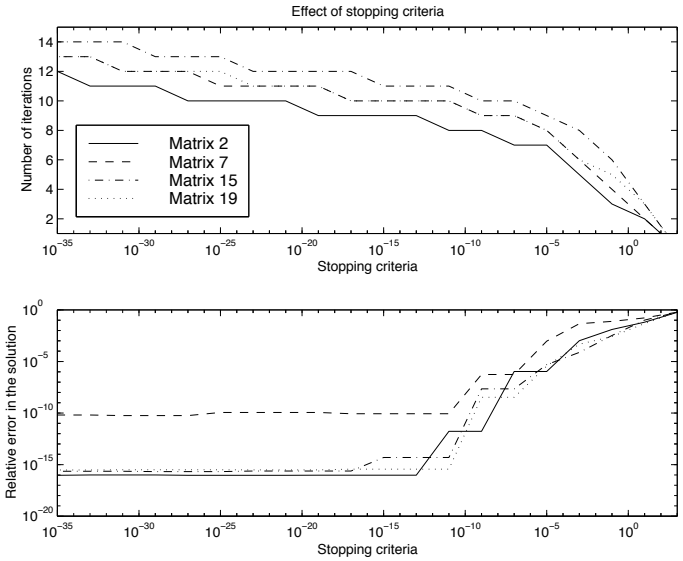

Figure 6.3: Number of iterations and relative error as a function of the stopping criteria ε_1. The test problems was nondegenerate.

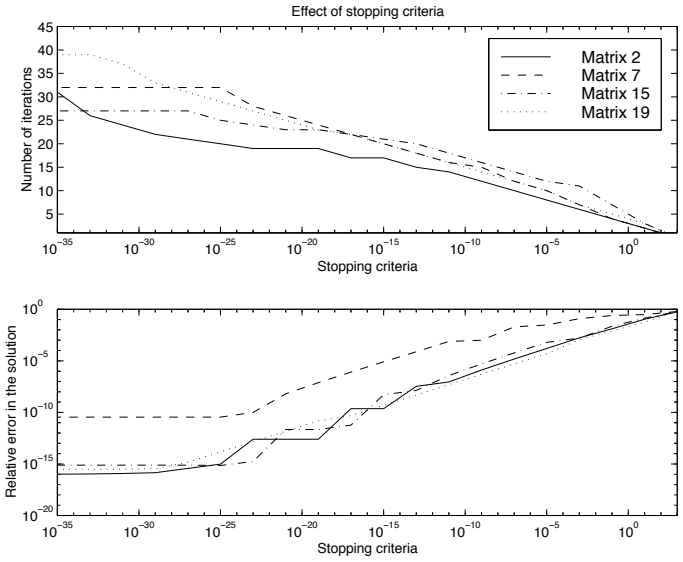

Figure 6.4: Number of iterations and relative error as a function of the stopping criteria ε_1. The test problems was degenerate.

the path and very near the solution. Partition the solution into two different sets, the free and bounded at the solution $x^T = (x_{\mathcal{F}}^T \; x_{\mathcal{B}}^T)^T$. Partition A and D likewise and observe that we now have $D = T^{\perp 1}V$. Then we can write the least squares problem (6.29) as

$$\min_{dx} \left\| \begin{pmatrix} A_{\mathcal{F}} & A_{\mathcal{B}} \\ D_{\mathcal{F}}^{1/2} & 0 \\ 0 & D_{\mathcal{B}}^{1/2} \end{pmatrix} \begin{pmatrix} dx_{\mathcal{F}} \\ dx_{\mathcal{B}} \end{pmatrix} - \begin{pmatrix} b_1 \\ b_{21} \\ b_{22} \end{pmatrix} \right\|_2^2 .$$

To analyze this closer we split this into three different parts,

$$\begin{aligned} \min_{dx} \Bigg\{ & ||A_{\mathcal{F}} dx_{\mathcal{F}} + A_{\mathcal{B}} dx_{\mathcal{B}} - (b - Ax)||_2^2 \\ & + \left\| D_{\mathcal{F}} \left(dx_{\mathcal{F}} - \frac{T_{\mathcal{F}}^{\perp 1} V_{\mathcal{F}} (t_{\mathcal{F}} + l_{\mathcal{F}} - x_{\mathcal{F}})}{T_{\mathcal{F}}^{\perp 1} V_{\mathcal{F}}} \right) \right\|_2^2 \\ & + \left\| D_{\mathcal{B}} \left(dx_{\mathcal{B}} - \frac{T_{\mathcal{B}}^{\perp 1} V_{\mathcal{B}} (t_{\mathcal{B}} + l_{\mathcal{B}} - x_{\mathcal{B}})}{T_{\mathcal{B}}^{\perp 1} V_{\mathcal{B}}} \right) \right\|_2^2 \Bigg\}. \end{aligned} \quad (6.30)$$

Consider the last term of (6.30). Since we are close to convergence the difference $t + l - x$ is close to zero, and since the solution is nondegenerate, $V_{\mathcal{B}}$ is bounded away from zero. Assuming the current iterate is close to the central path, the elements in $D_{\mathcal{B}} \approx V_{\mathcal{B}}^2/\mu$ diverge to ∞ when $\mu \to 0^+$. If the iterate is on the path the term $t + l - x$ will vanish). The term $dx_{\mathcal{B}}$ will therefore make the difference $t_{\mathcal{B}} + l_{\mathcal{B}} - x_{\mathcal{B}}$ as small as possible, otherwise the residual of this term will dominate the minimization problem.

If we now study the middle term, $T^{\perp 1}$ is bounded away from zero but $V \to 0^+$ and therefore $D_{\mathcal{F}} \approx T^{\perp 2}\mu \to 0^+$. This term will not influence the solution much if we are close to convergence. Observe that the correction dt is computed in each iteration to make t satisfy $t + l - x = 0$ and therefore the difference $t + l - x$ will be very close to zero near convergence.

Since the last term will influence the solution of $dx_{\mathcal{B}}$, and make this term very close to zero, the remaining variables, $x_{\mathcal{F}}$, will be governed by the first term. If we look at iterative improvement for the unconstrained problem when the bounded variables are fixed at their bounds we get,

$$\begin{gathered} r = (b - A_{\mathcal{B}} x_{\mathcal{B}}) - A_{\mathcal{F}} x_{\mathcal{F}} = b - Ax, \\ \min_{dx} ||Adx - r||_2, \\ x \leftarrow x + dx. \end{gathered}$$

This is very close to the first term when $dx_{\mathcal{B}}$ is suppressed by the last term of (6.30). The iteration will be very similar to iterative refinement close to the solution and therefore it will not be necessary to carry out any further iterative improvement to increase the accuracy of the solution.

Table 6.4: Condition number $\kappa(A)$ and scaled condition number $\kappa(\hat{A})$ during the iterations for two nondegenerate problem.

	ash219		ILLC1033	
It	$\kappa(A)$	$\kappa(\hat{A})$	$\kappa(A)$	$\kappa(\hat{A})$
0	7.8	2.1	$1.9 \cdot 10^4$	$1.9 \cdot 10^4$
1	$1.4 \cdot 10^2$	2.7	$4.4 \cdot 10^1$	$1.4 \cdot 10^1$
2	$4.2 \cdot 10^1$	3.7	$6.4 \cdot 10^7$	8.6
3	$1.7 \cdot 10^2$	5.8	$5.1 \cdot 10^2$	$2.6 \cdot 10^1$
4	$3.6 \cdot 10^9$	7.6	$1.0 \cdot 10^9$	$1.5 \cdot 10^2$
5	$5.2 \cdot 10^{10}$	7.7	$2.9 \cdot 10^8$	$1.4 \cdot 10^2$
6	$3.5 \cdot 10^{10}$	7.7	$5.7 \cdot 10^8$	$3.0 \cdot 10^3$
7	$7.2 \cdot 10^9$	7.7	$4.5 \cdot 10^{10}$	$2.3 \cdot 10^5$
8	$2.4 \cdot 10^{11}$	7.7	$4.0 \cdot 10^{14}$	$8.6 \cdot 10^5$
9	$2.7 \cdot 10^{19}$	7.7	$2.9 \cdot 10^{14}$	$1.0 \cdot 10^6$
10			$5.2 \cdot 10^{18}$	$1.1 \cdot 10^6$

The condition number of the matrix $(A^T D^{1/2})^T$ will become very large when $\mu \to 0^+$. However, this will not give inaccurate search direction, since the large condition number arises from badly scaled columns in A. If we rescale the columns to have unit norm, the condition number will not increase as much as without the scaling, see Table 6.4. As pointed out in Section 2.2.2, the scaled condition number can be used in the error analysis of the Cholesky factorization.

In our implementation we solve the least squares problems by solving the normal equations with the Cholesky factorization. One problem that could affect the performance is that when the small elements d_{ii} in D are less than $\sqrt{\mathbf{u}}||a_i||^2$ they will disappear when forming the normal equations due to round off. If we work with the QR factorization we have the square root of d_{ii} and the terms would therefore influence the solution of the least squares problem longer. However we did not notice any difference at all during the iterations using the normal equations instead of QR when solving nondegenerate problems.

We observed that when the small elements in D became smaller than $\sqrt{\mathbf{u}}||a_i||^2$ we did not get any improvement of the solution by either the normal equation or QR method. This is understandable because the small elements in D affect are just a scaling of μ by the Lagrange multipliers associated with this variable when close to convergence. So in essence when the elements d_{ii} are small enough to be in danger of round off, we have already converged. We did also try this with Lagrange multipliers of different magnitude and all results was the same. Note it is important that the right hand side is computed $A^T(b - Ax)$ and not $A^Tb - A^TAx$ for this reasoning to be justified.

For the degenerate case we could expect a greater difference between using the normal equations and the QR method since we have to iterate longer to

obtain the same accuracy as for the nondegenerate problem. However, we noted that when the small elements d_{ii} were smaller than $\sqrt{\mathbf{u}}||a_i||^2$ it took only two or three iterations until they was smaller than $\mathbf{u}||a_i||^2$ and could not influence the solution even when using the QR method. So in the last iterations the elements were so small that it did not matter if the normal equations or the QR method was used. This was also verified numerically, the solution by using QR instead of normal equations was not significantly better. Note however that the condition number of the test problems were all less than $2 \cdot 10^4$. As is well known the Cholesky algorithm may fail, unless $2n^{3/2}\mathbf{u}\kappa^2(A) < 0.1$. However our observation is consistent with that of Foster [18] and Björck [5, sec. 6.5.5] that if A is not very ill-conditioned then the Cholesky factorization with iterative refinement will just be as accurate as QR methods.

The sparsity pattern of $A^TA + D$ (6.29) is constant in each iteration and therefore we can compute the column ordering in the beginning and allocate memory for the R factor from the QR or Cholesky factorization. This will reduce computational cost, since the cost of the symbolic analysis phase can be of the same order of magnitude as the cost of the numerical factorization phase, see Matstoms [49].

If we initially compute the R factor by a QR factorization we may be able to reduce the cost for computing the solution to the least squares problem by replacing A by R and factorize $(R^T D^{1/2})^T$ instead. This approach was investigated by Matstoms [49] who concluded is that for the multi-frontal QR method we get better efficiency only if the nonzero count for R is smaller than A.

6.3 The rank deficient case

If the matrix A is rank deficient there may be multiple solutions to problem BLS as shown in Section 5. The question arises: which solution does the interior-point method converge toΓ In Lemma 6.1 we show that if the solution set is not bounded then the central path is not bounded and the method will diverge to the solution at ∞.

LEMMA 6.1 *The central path $\Gamma(\mu)$ defined by (6.10) is bounded if and only if the solution set to (6.3) is bounded.*

Proof. We will prove both directions by indirect proofs. First assume that $\Gamma(\mu)$ is not bounded for any μ. We show that then the solution set to (6.3) is also unbounded. If $\Gamma(\mu)$ is not bounded for any μ, then there exist a vector v such that $B(x + \alpha v, \mu) \to -\infty$, $\alpha > 0$ and $x + \alpha v$ is feasible for all $\alpha > 0$. Then $v \in \mathcal{N}(A)$ because $v^T A^T A v$ grows faster than the corresponding logarithmic terms. However, if $v \in \mathcal{N}(A)$ and $x + \alpha v$ is feasible, then the solution set to (6.3) is also unbounded. This shows that if the solution set to (6.3) is bounded then the path $\Gamma(\mu)$ is bounded.

Now assume the solution set is not bounded. We show that $\Gamma(\mu)$ is not bounded for any μ. If the solution set to $B(x, 0)$ is not bounded then there exist

a vector $v \in \mathcal{N}(A)$ so $x^* + \alpha v$, $\alpha > 0$ and where x^* is a solution to (6.3). Then clearly $B(x, \mu) \to -\infty$ when $\alpha \to \infty$ and $\Gamma(\mu)$ is not bounded for any μ. This concludes the proof. □

The interior-point method will not converge to a solution if the solution set of (6.3) is not bounded. If the solution set consists of more than one point it can often be of interest to obtain the minimum norm solution. Will the interior-point method generate this solutionΓ With the following counter example we will show that the minimum norm solution is not obtained by the interior-point methods. Suppose $\mathcal{N}(A) = e_1$ and that $0 \leq x_1 \leq 1$. Then the minimum norm solution will have $x_1 = 0$, but the solution from the interior-point method will converge to $1/2$[7].

A way of modifying the interior-point method so that it converges to the minimum norm solution, is to add a smoothing (regularization) term. The smoothing term will modify the matrix A to have full column rank and then the problem has a unique solution. A well known method is Tikhonov regularization,

$$\min_x ||Ax - b||_2^2 + \lambda ||x||_2^2.$$

This can be reformulated as,

$$\min_{x \in \mathcal{S}} \left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} dx - \begin{pmatrix} b_1 \\ 0 \end{pmatrix} \right\|_2. \tag{6.31}$$

This problem has the same structure as systems solved in the interior point methods and is easy to implement since it will not alter the nonzero structure of R in the solution of the unconstrained least squares problems.

The unconstrained solution of this problem is

$$x(\lambda) = \sum_{i=1}^{n} \frac{\sigma_i u_i^T b}{\sigma_i^2 + \lambda^2},$$

which converges to the pseudo-inverse solution as $\lambda \to 0^+$. Here σ_i and u_i are the singular values and the left singular vectors to A.

A difficulty is to choose the regularization parameter λ. If λ is too large we loose accuracy in the solution and if λ is chosen too small the method will fail to converge to the minimum norm solution. In Figure 6.5 the typical behavior of the central path when the regularization parameter is added is shown. When μ is large the regularization parameter does not influence the solution path but when the elements in $D_{\mathcal{F}} < \lambda$ the regularization parameter affects the solution and the iterates converges to the minimum norm solution. If we choose λ really small there will be a sharp corner on the path Γ.

It is always difficult to choose the right regularization parameter. If our problem is not "ill-conditioned" in the sense that we have small singular values due to noise and other disturbances in data, we would like to have λ as small as

[7] This result is obtained by minimizing the barrier function respect to x_1

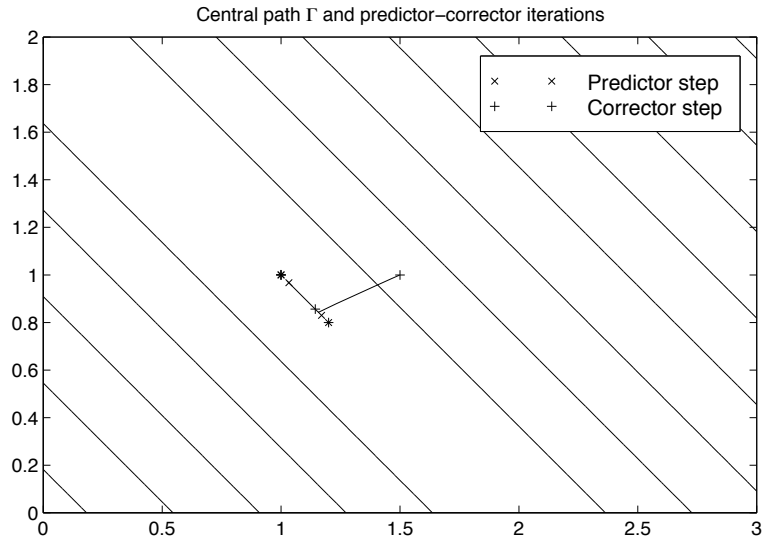

Figure 6.5: Regularized solution path and the interior-point iterations, regularization parameter $\lambda = 10^{\perp 4}$.

possible. If λ is chosen too large we will get an error in the solution due to the regularization. If λ is chosen too small it will not influence the solution at all, since the interior-point method will converge to the wrong solution before the regularization affects the solution. What we want is an adaptive way to decrease the regularization parameter as the iterates tends to the solution. Since there are already some heuristic features in the interior-point method, it does not feel awkward to add an adaptive choice of λ. We have had some success by choosing the regularization parameter as,

$$\lambda = \min\left(10^{\perp 4},\ \mu^{1/4}\right).$$

We have using a larger exponent than $1/4$, but then the regularization parameter gets too small and does not influence the solution enough to yield the minimum norm solution.

7 Numerical comparison

In this section we make a comparison between the two different approaches presented in Section 5 and 6 for solving the least squares problem with box constraints. We will choose one active set method and compare it against one of the interior-point methods. We use our standard test problem suite introduced in section 4. The comparison was carried out on a Sun Ultra Sparc 2200, 200 MHz and 1 GB primary memory using MATLAB 5.1.0. The first comparison we make is with the test problems denoted as type A and B in section 4.2. Thereafter we present another test problem suite to compare how the different methods deal with different numbers of free variables at the solution.

7.1 Numerical results

The active set method we have chosen to be our candidate in this comparison is the Algorithm 5.4, denoted `BLOCK3`. This method was slightly slower than the `BLOCK2` method, Algorithm 5.3, in general. However, we feel that this method is more stable than `BLOCK2` since we have a convergence proof for the method and we do not have to use a single pivoting mechanism to ensure convergence. The optimal point is assumed to be detected when all variables are feasible and the Lagrangian is positive with a tolerance of $n\mathbf{u} \cdot 10^2$, where n is the number of variables. When the active set is found the solution is recomputed with a QR factorization to obtain higher accuracy.

For the interior-point method, denoted `IP` hereafter, we used one correction since it was only for the largest matrix more corrections gave any improvement in the MATLAB implementation. The intermediate least squares problems were solved with a Cholesky factorization of A^TA. However, since the Cholesky factorization may fail when A is badly conditioned, a safeguard was implemented to carry out a QR factorization if the Cholesky factorization failed. The Cholesky factorization did not fail for the test problems here, but in our experience we did encounter other problems where the Cholesky factorization failed. In particular, this can happen for rank deficient problems with an added regularization term, see Section 6.3.

In the interior-point method we used different tolerances in the stopping criteria for the nondegenerate and degenerate problems to be able to obtain the same accuracy of the solution. In the nondegenerate case, type A problems, we used $\varepsilon_1 = 10^{\perp 18}$ and in the degenerate case, type B problems $\varepsilon_1 = 10^{\perp 25}$ was used. The second tolerance, ε_2 was set to $10^{\perp 12}$ in both cases. When the interior-point method terminates, the solution is not recomputed with a QR factorization. The minimum degree ordering was used for all matrices to reduce the fill-in in the R factor.

In Table 7.1 we have compared the number of factorizations and the computational time for solving the test problems. For the nondegenerate test problems the `BLOCK3` was superior for most of the problems. We can note that it was only in the more ill-conditioned problem 7 and 13 `BLOCK3` used more factorizations than `IP`. However, only for problems 13 and 14 was `BLOCK3` slower than `IP`.

In problem 14 BLOCK3 is slower due to the last resolving phase. However, the accuracy drops significantly if the QR factorization is not used.

In Figure 7.1 the relative error of the solution measured in the 2-norm is shown. We see here, surprisingly, that for the nondegenerate problems the interior-point method yields a more accurate solution that the active set method. However, this may not be entirely true, since the error in the reference solution is of the same magnitude as the error shown in the plot. For the degenerate problems we have a larger difference in the accuracy between the two methods. The accuracy of the interior-point method is one to two magnitudes worse than for the active set method.

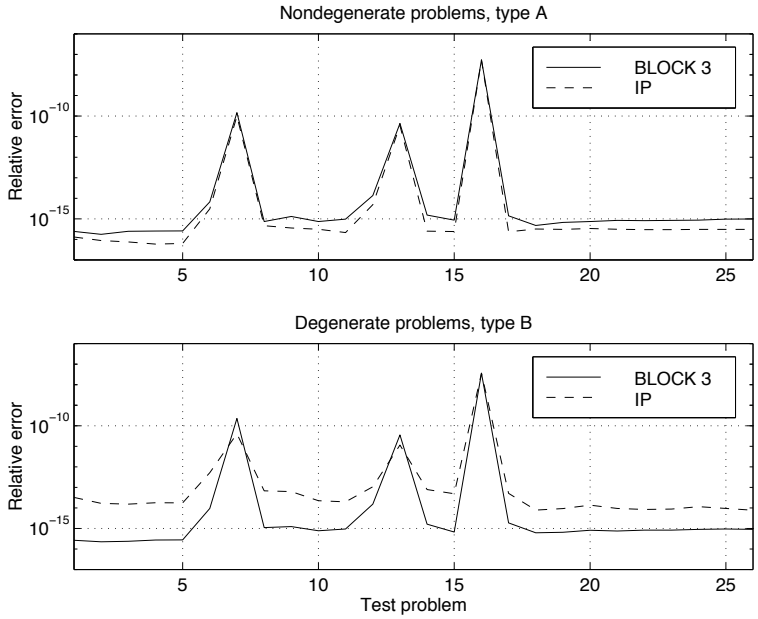

Figure 7.1: Comparison of the relative error in the solution for the BLOCK3 and IP.

An alternative we considered was to make a hybrid algorithm, by combining the rapid convergence of the interior-point algorithm in the beginning with the active set method. This can be done by using a weaker stopping criteria in IP and then switching to the active set method by identifying the active set and using this information as a warm start. However, we found that we did not gain any computational effort using this hybrid scheme instead of using the active set method from the beginning. In the first few iterations with the active set the method finds most of the members in the different sets, and the last iterations only make fine adjustments to the sets. Computational effort could be gained compared to the interior-point method with this scheme, but we would loose one of the attractive features of the interior-point method, namely that a

Table 7.1: Comparison of `BLOCK3` against Mehrotra's interior-point method, `IP`. Left table shows number of factorizations and the right table shows the computational time in seconds.

	Nondeg., A		Deg., B	
Prob.	B3	IP	B3	IP
1	4	10	4	32
2	4	9	3	32
3	3	9	3	32
4	3	8	3	32
5	4	9	3	32
6	5	16	6	33
7	13	10	18	35
8	6	9	6	32
9	5	9	5	32
10	3	8	3	32
11	3	9	3	32
12	7	9	7	33
13	13	11	22	37
14	5	9	6	32
15	8	12	8	35
16	9	13	16	36
17	9	15	9	40
18	4	10	4	33
19	5	11	4	33
20	5	11	5	33
21	5	11	5	33
22	5	11	5	33
23	5	11	5	33
24	5	11	5	33
25	5	11	5	33
26	5	11	5	33

	Nondeg., A		Deg., B	
Prob.	B3	IP	B3	IP
1	0.12	0.41	0.16	1.12
2	0.07	0.19	0.05	0.57
3	0.05	0.22	0.06	0.66
4	0.12	0.25	0.11	1.11
5	0.17	0.42	0.15	1.44
6	0.28	0.94	0.35	1.56
7	0.44	0.54	1.18	1.81
8	0.51	0.67	0.48	1.95
9	0.53	0.68	0.49	2.08
10	0.72	0.63	0.75	2.60
11	1.11	1.29	1.07	4.04
12	0.73	1.00	0.82	3.80
13	1.51	1.41	2.63	4.36
14	3.46	2.55	3.41	8.50
15	1.57	2.62	1.51	7.79
16	5.64	15.5	13.7	41.6
17	37.8	53.9	48.7	140
18	0.10	0.25	0.08	0.72
19	0.36	0.98	0.36	2.71
20	1.07	2.12	1.18	6.35
21	2.12	4.76	2.4	13.93
22	4.18	8.11	4.2	28.48
23	6.47	15.0	7.02	59.2
24	9.38	24.0	9.75	82.7
25	15.0	35.0	15.3	112
26	20.0	44.5	20.8	147

system of linear equation with the same structure is solved in all iterations. In our MATLAB implementation this could not be exploited in the same manner as for example in a Fortran code. We suspect that the time difference would not be as great if the methods are implemented with a real programming language, since then the analysis phase of the factorizations need only be carried out once. (The analysis phase can be as time consuming as the factorizations phase [49].)

The third test is made to determine if the methods are sensitive in any way to the number of active constraints at the solution. The interior-point method should not be influenced in the same manner as the active set method, since we always approach the solution from the interior of the feasible domain. The in Table 7.2 we have used the same parameters that define the nondegenerate type A problem above, except for the distribution of the variables in the different sets. The free set, $\mathcal{F}$ consists of $n \cdot (1.0,\ 0.8,\ 0.6,\ 0.4,\ 0.2,\ 0)$ free variables and the remaining are evenly divided between the lower and upper bound. We conclude from Table 7.2 that the interior-point method is not affected much by the different size of the free set. However, the active set method is affected and converges most slowly when the number of free and bounded variables are nearly equally balanced. We can also conclude from test problems 18–26 in Table 7.1, that the size of the problem does not affect the number of iterations for either method, when the structure is the same. In Table 7.3 the computational time is shown for the problems in Table 7.2. Note that for problem 15 the computational time of the active set method for the problem with all variables free is greater than when the problem has 60% free variables or less. This is due to the last QR factorization is dominant in the computation. The factorization is very time consuming compared to the Cholesky factorization and when the index set of free variables is small, the QR factorization will not be as dominant. This behavior was not observed if the last QR step was omitted. Then the active set method was always fastest for the "unconstrained" problem.

Table 7.2: Number of factorizations for `BLOCK3` and Mehrotra's interior-point method with different sizes of the free set, $\mathcal{F}$. Problem type A.

Density	1.0		0.8		0.6		0.4		0.2		0	
Prob.	B3	IP	B3	IP	B3	IP	B3	IP	B3	IP	B3	IP
2	1	7	3	10	4	10	5	10	4	10	4	10
6	1	10	7	9	6	9	6	10	7	10	4	12
15	1	9	8	11	8	12	9	13	7	14	6	13
20	1	8	5	11	5	11	5	11	5	11	4	12

If we compare the memory usage for both methods, it is essentially the same. For both methods the largest memory requirements is for the R factor and (when used) for the matrix A^TA. In the interior-point method we store A^TA throughout the iterations. In the active set method, $A_{\mathcal{F}}^TA_{\mathcal{F}}$ is generated in each iteration from a subset of the columns of A. Except for R and A^TA only a few $n \times 1$ vectors are used, where n is the number of variables. For the

Table 7.3: Computational time for BLOCK3 and Mehrotra's interior-point method with different sizes of the free set, $\mathcal{F}$. Problem type A.

Density	1.0		0.8		0.6	
Prob.	B3	IP	B3	IP	B3	IP
2	0.03	0.12	0.05	0.17	0.05	0.17
6	0.15	0.53	0.34	0.47	0.29	0.48
15	2.17	2.29	2.40	2.43	1.81	2.64
20	1.02	1.48	1.49	2.01	1.17	2.08

Density	0.4		0.2		0	
Prob.	B3	IP	B3	IP	B3	IP
2	0.06	0.17	0.05	0.17	0.04	0.17
6	0.28	0.52	0.24	0.53	0.14	0.62
15	1.59	2.88	0.73	3.13	0.63	2.96
20	0.88	2.09	0.57	2.27	0.40	2.75

interior-point method we use about 16 vectors and in the active set method 13. These vectors use storage of the same order of magnitude as the A factor. We could easily reduce the storage needed for the interior point method from 16 to 9 vectors by sacrificing some efficiency. However, since storage is cheap compared to computational speed we did not feel obliged to reduce the memory usage. If the algorithm would be implemented with a real programming language, we would regard this question with a higher importance.

If we have a series of related problems where the solution changes a little for each problem, we recommend the active set method before the interior-point method. The active set method is very easy to restart and if the index set of the free variables is the same we would get the solution in one iteration. When using the interior-point method we would gain some iterations, but we would still need a lot of iterations to obtain high accuracy. The behavior in a restart can in some sense be compared with solving the "unconstrained" problem with the methods. For that problem we still need some iterations to obtain high accuracy for the interior-point method.

8 Conclusion

We have studied two different type of methods for solving the least squares problem with box constraints. Both approaches uses direct methods, the QR method or the method of normal equations, when solving each unconstrained least squares problem.

In the first approach we used active set methods. The single pivot algorithm, that moves one variable between the sets, was generalized in different ways to more effective block algorithms that move several variables in each iteration. We tried four different block methods and concluded that all algorithms were very efficient and accurate. For most problems the most efficient scheme was a heuristic approach proposed by Portugal et. al. [62]. However, for this algorithm to be effective in general, an updating technique of R when adding (or deleting) a column of A should be implemented to handle the algorithm in the single pivoting mode. We gave a convergence proof for an algorithm called `BLOCK3` that uses a backward search to ensure reduction of the objective function in each step. This algorithm was the candidate in the final comparison.

In the second approach we applied an interior-point method proposed by Mehrotra [51] to our problem. We tried several different heuristics for determining the important centering parameter μ and concluded that the initial suggestion by Mehrotra was the best choice. Further, we tried to use each factorization several times by using a multiple correction scheme. We saw a clear reduction of the number of factorizations needed when using such a scheme. However, in our MATLAB iimplementation we did not gain any computational time by this except for the largest degenerate problem where a small gain was noticed. However, if an implementation is made in another programming language, for example Fortran or C, we would expect improvement in computational time when using three to five corrections in each step.

We concluded for problems which was not too ill-conditioned the method of normal equations gave solutions of high accuracy This was explained by noticing that the interior-point method behaves as an iterative refinement scheme in the last iterations. We showed how to solve rank deficient problems with the interior-point method by adding a regularization term and letting the regularization parameter tend to zero as the barrier parameter $\mu \to 0^+$.

In the final comparison of the two approaches we clearly saw that the active set method was more efficient than the interior-point method, except for two problems. The active set method was 2-10 times faster than the interior-point method, the higher figure is valid for degenerate problems. The accuracy of the solution is comparable for both types of methods in the nondegenerate case. For the degenerate case we obtained better accuracy with the active set method.

8.1 Further research

We list below some interesting points that deserve further studies.

- Methods for problems with more general constraints on the variables, $Cx \leq d$, need to be developed. In particular problems where C has a

sparse structure, for example banded structure.

- In the active set methods it would be interesting to see if efficient updating is possible using the information from the multi-frontal algorithm. There has been some recent development for modifying the Cholesky factorization of A^TA by adding/removing rows to A [11]. A similar scheme may be applicable to the multi-frontal QR algorithm.
- If the matrix A has a full row then A^TA is a full matrix and all sparsity is destroyed. However, there are some techniques for handling a few full rows and there could easily be implemented into the solvers.
- A lot of effort remains in developing methods to solve the rank deficient case efficiently. The behavior of methods adding a regularization term in the interior-point method should be studied more thoroughly.
- Iterative methods to solve the least squares problems has been used in the active set method by Lötstedt [42]. This approach could turn out to be successful in the interior-point method also.

A Matlab m-files

sbls

Purpose

Solve sparse least squares with box constraints.

Synopsis

`x=sbls(A,b,l,u)`

Description

`x=sbls(A,b,l,u)` solves the constrained sparse linear least squares problem,

$$\min_x ||Ax - b||_2, \text{ subject to } l \leq x \leq u$$

where A is a sparse m -by- n matrix.

Example

Generate a sparse test problem and calculate the relative error of the solution.

```
>> [A,b,l,u,xexact] = sblsgen (300,50,0.008,1000,0,0,100,100);
>> x = sbls (A,b,l,u);
>> norm(xexact-x)/norm(xexact)

ans =

   3.7275e-17
```

Algorithm

The solution x is computed by an predictor-corrector path following interior-point method introduced by Mehrotra [1].

References

[1] S. Mehrotra. *On Finding a Vertex Solution Using Interior Point Methods.*

sbls2

Purpose

Solve sparse least squares with box constraints.

Synopsis

```
x=sbls2(A,b,l,u)
```

Description

`x=sbls2(A,b,l,u)` solves the constrained sparse linear least squares problem,

$$\min_x ||Ax - b||_2, \text{ subject to}$$
$$-\infty \leq l_i \leq x_i \leq u_i \leq \infty$$

and A is a sparse m -by- n matrix.

Example

Generate a sparse test problem and calculate the relative error of the solution.

```
>> [A,b,l,u,xexact] = sblsgen (300,50,0.008,1000,0,0,100,100);
>> x = sbls2 (A,b,l,u);
>> norm(xexact-x)/norm(xexact)

ans =

   7.5802e-17
```

Algorithm

The solution x is computed by a block active set method, described by [1].

References

[1] L. F. Portugal et. al. *"A comparison of block pivoting and interior-point algorithms for linear least squares problem with nonnegative variables.*

quadres

Purpose

Compute a residual, $b - Ax$, using quadruple precision.

Synopsis

`res=quadres(A,x,b)`

Description

`res=quadres(A,x,b)` computes the residual, $res = b - Ax$, using quadruple precision instead of MATLAB's usual IEEE double precision. This is done by a C mex file `quadres.c`. To compile this file type: `cmex spres.c`, if the architecture do not support quadruple precision, double precision is used instead.

Computing a residual with higher precision is useful when doing iterative refinement.

Example

Generate a sparse test problem and calculate the residual using MATLAB commands and with `quadres`

```
>> A=sprandn(400,100,0.01);
>> x=ones(100,1);
>> b=A*x;
>> norm(b-A*x)

ans =

     0

>> norm(quadres(A,x,b));

ans =

     8.326672684688674e-17
```

B Residual computation in quadruple precision

```
/* Residual in quadruple precision for Matlab.
   r=quadres(A,x,b),  calculates  r = b - Ax in quadratic precision,
   A is a sparse or dense matrix, b and x are dense vectors.
   Compile with a compiler that supports long doubles or extended
   reals.

   Compile with: mex quadres.c
             or: cc -c -I$MATLAB/extern/include quadres.c
                 cmex quadres.o
   where $MATLAB refers to the dir where Matlab is installed.
   MATLAB V5 assumed.

   @(#)quadres.c Version 1.5 8/15/97
   Mikael Adlers, Department of Mathematics
   Linkoping University.
   e-mail: milun@mai.liu.se
*/
#include "mex.h"

#define A 0
#define x 1
#define b 2
#define res 0

/* --------------- Sparse matrix residual ---------------*/

#ifdef __STDC__
  void spres(mxArray *Am, mxArray *xm, mxArray *bm, mxArray *pout)
#else
  spres(Am,xm,bm,pout)
  mxArray *Am,*xm,*bm,*pout
#endif
{
long double *ldA,*ldx,*ldb, *ldres;
int M,N,nnz,i,j;
int *jc,*ir;
double *db, *dx, *dA,*dout;
long double xx;

/* --- Extract pointer to the data --- */

M=mxGetM(Am);
N=mxGetN(Am);

db=mxGetPr(bm);
dx=mxGetPr(xm);
dout=mxGetPr(pout);

jc=mxGetJc(Am);
ir=mxGetIr(Am);
dA=mxGetPr(Am);
nnz=mxGetNzmax(Am);

/* --- Create storage for the long doubles and move data --- */
```

```
ldA=mxCalloc(nnz,sizeof(long double));
ldx=mxCalloc(N,sizeof(long double));
ldb=mxCalloc(M,sizeof(long double));
ldres=mxCalloc(M,sizeof(long double));

for (i=0;i<nnz;i++)  {ldA[i]=dA[i];}
for (i=0;i<N;i++)    {ldx[i]=dx[i];}
for (i=0;i<M;i++)    {ldb[i]=db[i];}

/* --- Perform the computation column wise --- */
/* res=Ax */
for (j=0;j<N;j++){
    xx=ldx[j];
    for (i=jc[j];i<jc[j+1];i++){
      ldres[ir[i]]=ldres[ir[i]]+ldA[i]*xx;}
  }

/*res = b-res */
for (i=0;i<M;i++){
    ldres[i]=ldb[i]-ldres[i];}


/* --- Move data back to output vector --- */
for (i=0;i<M;i++){
    dout[i]=(double)ldres[i];}

/* --- clear memory space --- */
mxFree(ldA);mxFree(ldx);mxFree(ldb);mxFree(ldres);
}

/* --------------- Dense matrix residual ---------------*/

#ifdef __STDC__
  void FullRes(mxArray *Am, mxArray *xm, mxArray *bm, mxArray *pout)
#else
  FullRes(Am,xm,bm,pout)
  mxArray *Am,*xm,*bm,*pout
#endif
{
long double *ldA,*ldx,*ldb, *ldres;
int M,N,ind,i,j;
double *db, *dx, *dA,*dout;
long double xx;

/* --- Extract pointer to the data --- */

M=mxGetM(Am);N=mxGetN(Am);

dA=mxGetPr(Am);
db=mxGetPr(bm);
dx=mxGetPr(xm);
dout=mxGetPr(pout);

/* --- Create storage for the long doubles and move data --- */

ldA=mxCalloc(M*N,sizeof(long double));
ldx=mxCalloc(N,sizeof(long double));
```

```
ldb=mxCalloc(M,sizeof(long double));
ldres=mxCalloc(M,sizeof(long double));

for (i=0;i<M*N;i++)  {ldA[i]=dA[i];}
for (i=0;i<N;i++)    {ldx[i]=dx[i];}
for (i=0;i<M;i++)    {ldb[i]=db[i];}

/* --- Perform the computation column wise --- */
/* res=Ax */
ind=0;
for (j=0;j<N;j++){
    xx=ldx[j];
    for (i=0;i<M;i++){
ldres[i]=ldres[i]+ldA[ind]*xx;
ind++; }
  }

/*--- res = b-res --- */
for (i=0;i<M;i++){
    ldres[i]=ldb[i]-ldres[i];}


/* --- Move data back to output vector --- */
for (i=0;i<M;i++){
    dout[i]=(double)ldres[i];}

/* --- Clear memory space --- */
mxFree(ldA);mxFree(ldx);mxFree(ldb);mxFree(ldres);
}


/* --------------- Entry point --------------- */
#ifdef __STDC__
void mexFunction(
 int nlhs,
 mxArray *plhs[],
 int nrhs,
 const mxArray *prhs[]
 )
#else
  mexFunction(nlhs,plhs,nrhs,prhs)
  int nlhs, nrhs;
  const mxArray *plhs[], *prhs[];
#endif
{
  unsigned int m1,n1,m2,n2,m3,n3;
  double *t, *y;
  double *pr;
  int nzmax,*ir,*jc;
  double k;


/* --- Check that the input parameters are correct --- */

  if (nrhs != 3)
    {mexErrMsgTxt("Three input arguments required.");}
  else if (nlhs > 1)
```

```
    {mexErrMsgTxt("Only one output arguments.");}

  m1=mxGetM(prhs[A]);  n1=mxGetN(prhs[A]);
  m2=mxGetM(prhs[x]);  n2=mxGetN(prhs[x]);
  m3=mxGetM(prhs[b]);  n3=mxGetN(prhs[b]);

  if (( n2 != 1)||(n3 != 1)||(m2!=n1)||(m3!=m1))
    {mexErrMsgTxt("Input dimensions are wrong");}

  if(mxIsComplex(prhs[A]) || !mxIsDouble(prhs[A]))
    {mexErrMsgTxt("A must be a real numeric matrix.");}


/* --- If the compiler supports long double but defines them as doubles --- */
/* --- use MATLAB to compute the residual --- */
  if (sizeof(long double) != 16) {
      mxArray *inpt[2];
      mxArray *outp;

      mexPrintf("WARNING!!! This architecture or compiler do not handle
                 quadruple precision,");
      mexPrintf("this calculation will not occur in extended precision.\n");

      inpt[0]=prhs[A];
      inpt[1]=prhs[x];
      mexCallMATLAB(1,&outp,2,inpt,"*");
      inpt[1]=outp;
      inpt[0]=prhs[b];
      mexCallMATLAB(1,&outp,2,inpt,"-");
      mxDestroyArray(inpt[1]);
      plhs[0]=outp;
    }
  else {
      plhs[res]= mxCreateDoubleMatrix(m1,1,mxREAL);
      if (!mxIsSparse(prhs[A]))
  FullRes(prhs[A],prhs[x],prhs[b],plhs[res]);
      else
  spres(prhs[A],prhs[x],prhs[b],plhs[res]);
      }
}
```

References

[1] P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17:886–905, 1996.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, editors. *LAPACK Users's Guide – 2nd ed.* Society for Industrial and Applied Mathematics, Philadelphia, 1995.

[3] Å. Björck. Stability analysis of the method of semi-normal equations for least squares problems. *Linear Algebra Appl.*, 88/89:31–48, 1987.

[4] Å. Björck. A direct method for sparse least squares problems with lower and upper bounds. *Numer. Math.*, 54:19–32, 1988.

[5] Å Björck. *Numerical Methods for Least Squares Problems.* Society for Industrial and Applied Mathematics, Philadelphia, 1996.

[6] Å. Björck and C. C. Paige. Solution of augmented linear systems using orthogonal factorizations. *BIT*, 34:1–26, 1994.

[7] P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constraind problems. *Math. Programming*, 39:93–116, 1987.

[8] T. J. Carpenter, I. J. Lustig, J. M. Mulvey, and D. F. Shanno. Higher-order predictor-corrector interior point methods with application to quadratic objectives. *SIAM J. Optim.*, 3:696–725, 1993.

[9] T. F. Coleman and L. A. Hulbert. A direct active set algorithm for large sparse quadratic programs with simple bounds. *Math. Programming*, 45:373–406, 1989.

[10] T. F. Coleman and Y. Li. A reflective Newton method for minimizing a quadratic function subject to bounds on some ot the variables. Technical Report CTC92TR111, Department of Computer Science, Cornell University, 1992.

[11] T. Davis and W. Hager. Modifying a sparse Cholesky factorization. Technical report, Deptartment of Computer and Information Science and Engineering, University of Florida, 1997.

[12] R. S. Dembo and U. Tulowitzski. On the minimization of a quadratic function subject to box constraints. Working paper No. 71 Series B, School of Organization and Management, Yale University, 1983.

[13] Z. Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM J. Optim.*, 7:871–887, 1997.

[14] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, London, 1986.

[15] I. S. Duff, R. G. Grimes, and J. G. Lewis. User's guide for Harwell-Boeing sparse matrix test problems collection. Technical Report RAL-92-086, Rutherford Appleton Laboratory, 1992.

[16] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Software*, 9:302–325, 1983.

[17] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, Inc., New York, 2nd edition, 1987.

[18] L. V. Foster. Modifications of the normal equations method that are numerically stable. In G. H. Golub and P. Van Dooren, editors, *Numerical Linear Algebra, Digital Signal Processing and Parallel Algorithms*, NATO ASI Series, pages 501–512. Springer-Verlag, Berlin, 1991.

[19] C. F. Gauss. *Theory of the Combination of Observations Least Subject to Errors. Part 1, Part 2, Supplement,* G. W. Stewart, Trans. SIAM, Philadelphia, 1995.

[20] A. George. On finding and analyzing the structure of the Cholesky factor. In G. W. Althaus and E. Spedicato, editors, *Algorithms for Large Scale Linear Algebraic Systems*, pages 73–105. Kluwer Academic Publishers, 1998.

[21] J. A. George and M. T. Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra Appl.*, 34:69–83, 1980.

[22] J. A. George, M. T. Heath, and E. G. Ng. A comparison of some methods for solving sparse linear least squares problems. *SIAM J. Sci. Statist. Comput.*, 4:177–187, 1983.

[23] J. A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[24] J. A. George and J. W.-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31:1–19, 1989.

[25] J. R. Gilbert. *Graph Separator Theorems and Sparse Gaussian Elimination.* Ph.D. thesis, Stanford University, CA, 1980.

[26] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13:333–356, 1992.

[27] J. R. Gilbert, E. G. Ng, and B. W. Peyton. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 16:1075–1091, 1994.

[28] J. R. Gilbert, E. G. Ng, and W. Peyton. Separators and struckture prediction in sparse orthogonal factorization. *Linear Algebra Appl.*, 262:83–97, 1997.

[29] P. E. Gill and W. Murray. Numerically stable methods for quadratic programming. *Math. Programming*, 14:349–372, 1978.

[30] P. E. Gill, W. Murray, M. A. Sounders, and M. H Wright. Inertia-controlling methods for general quadratic programming. *SIAM Review*, 33(1):1–36, 1991.

[31] P. E. Gill and W. Murray M. H. Wright. *Numerical Linear Algebra and Optimization. vol 1.* Addison-Wesley Publishing Company, Inc., 1991.

[32] G. H. Golub. Numerical methods for solving least squares problems. *Numer. Math.*, 7:206–216, 1965.

[33] G. H. Golub and C. F. Van Loan. *Matrix Computations. 3nd ed.* The John Hopkins University Press, Baltimore, 1989.

[34] P. Heggernes and P. Matstoms. Finding good column orderings for sparse QR factorization. Technical Report LiTH-MAT-R-1996-20, Departement of Mathematics, Linköpings University, 1996.

[35] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.

[36] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *J. Assoc. Comput. Mach.*, 5:339–342, 1958.

[37] M. M. Kostreva. Block pivot methods for solving the complementarity problem. *Linear Algebra Appl.*, 21:207–215, 1978.

[38] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*, volume 15 of *Classics in Applied Mathematics.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.

[39] J. W.-H. Liu. On general row merging schemes for sparse Givens transformations. *SIAM J. Sci. Statist. Comput.*, 7:1190–1211, 1986.

[40] J. W.-H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11:134–172, 1990.

[41] J. W. H. Liu, E. G. Ng, and B. W. Peyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14:242–252, 1993.

[42] P. Lötstedt. Solving the minimal least squares problem subject to bounds on the variables. *BIT*, 24:206–224, 1984.

[43] S.-M. Lu and J. L. Barlow. Multifrontal computation with the orthogonal factors of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 17(3):658–679, 1996.

[44] I. Lustig, R. Marsten, and D. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra Appl.*, 152:191–222, 1991.

[45] I.J Lustig, R. E. Marsten, and D.F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM J. Optim.*, 2:435–449, 1992.

[46] O. L. Mangasarian. *Nonlinear Programming*. Number 10 in SIAM Classics in Applied Mathematics 10. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994.

[47] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

[48] P. Matstoms. Sparse QR factorization in MATLAB. *ACM Trans. Math. Software*, 20:136–159, 1994.

[49] P. Matstoms. *Sparse QR Factorization with Applications to Linear Least Squares Problems*. PhD thesis, Linköping University, 1994.

[50] P. Matstoms. Sparse linear least squares problem optimization. *Computational optimization and applications*, 7:89–110, 1997.

[51] S. Mehrotra. On finding a vertex solution using interior point methods. *Linear Algebra Appl.*, 152:233–253, 1991.

[52] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM J. Optim.*, 2:575–601, 1992.

[53] D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. part II. convex quadratic programming. *Math. Programming*, 44:43–66, 1989.

[54] R. D. C. Monteiro and I. Adler. Interior path following primal-dual algorithms. part I. linear programming. *Math. Programming*, 44:27–41, 1989.

[55] R. D. C. Monteiro, I. Adler, and M. G. C. Resende. A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming and its power series extension. *Math. Op. Res*, 15:191–214, 1990.

[56] J. J. Moré and G. Toraldo. Algorithms for bound constrained quadratic programming problems. *Numer. Math.*, 55:377–400, 1989.

[57] J. J. Moré and S. J. Wright. *Optimization Software Guide*. Number 14 in SIAM Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1993. ISBN 0-89871-322-6.

[58] K. Murty. Note on a Bard-type scheme for solving the complementarity problem. *Oper. Res.*, 11(123–130), 1974.

[59] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Number 13 in SIAM Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1994. ISBN 0-89871-319-6.

[60] U. Oreborn. *A Direct Method for Sparse Nonnegative Least Squares Problems*. Licentiat thesis, Linköping University, 1986.

[61] C. C. Paige. An error analysis of a method for solving matrix equations. *Math. Comp.*, 27:355–359, 1973.

[62] L. F. Portugal, J. Judice, and L. N. Vincente. A comparison of block pivoting and interior-point algorithms for linear least squares problem with nonnegative variables. *Math. Comp.*, 63:625–643, 1994.

[63] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, N.J., 1969.

[64] C. Sun. Dealing with dense rows in the solution of sparse linear least squares problems. Technical report, Cornell Theory Center, Cornell University, 1995.

[65] T. Tapia, Y. Zhang, M. Saltzman, and A. Weiser. The Mehrotra predictor-corrector interior-point method as a perturbed composite Newton method. *SIAM J. Optim.*, 6(1):47–56, 1996.

[66] W. F. Tinney and J. W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55:1801–1809, 1967.

[67] S. A. Vavasis. Quadratic programming is in NP. *Inform. Process. Lett.*, 36:73–77, 1990.

[68] M. H. Wright. Interior methods for constrained optimization. In *Acta Numerica*, pages 341–407. Cambridge University Press, 1991.

[69] S. J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.