OPTIMIZATION METHODS BASED ON
PROJECTED VARIABLE METRIC SEARCH

J.F. BALLINTIJN, G. van der HOEK and C.L. HOOYKAAS

REPORT 7821/0

ERASMUS UNIVERSITY ROTTERDAM, P.O. BOX 1738, ROTTERDAM, THE NETHERLands
Summary.

As a first step to the realization of a new computer program to solve general nonlinear optimization problems, as a possible replacement of M.A.P. (Method of approximate programming, see Griffith and Stewart [12], we have developed a computer code which minimizes a nonlinear objective function subjected to a set of linear equality and/or inequality constraints. The method we have chosen is a generalization to linearly constrained problems of the variable metric technique upon which the well known algorithms for unconstrained optimization of Davidon [2], Fletcher and Powell [4], Broyden [1] and many others are based. This choice was based on the fact that quasi Newton (=variable metric) techniques compared very favorably with the optimization methods used in the past. We therefore expect the new algorithm to be faster and more robust than the algorithms dealing with uncorrected gradient information.

Part I of this report describes the mathematics and theoretical backgrounds behind our new linearly constrained optimization code, which we have called: VLICO (a Variable metric method for Linearly Constrained Optimization.)

In part II we discuss the extension of the linearly constrained optimization code VLICO, to the case of the general nonlinear programming problem. We have used for this extension the two phase method described by J.B. Rosen [22]. The resulting algorithm has been implemented in a computer program called VANOP (Variable metric Nonlinear Optimization) which has shown a fast and robust convergence behavior on a broad class of test problems, and therefore may be a possible replacement for the MAP code which is now often used.

1) Koninklijke Shell Laboratorium, Amsterdam
2) Econometric Institute, Erasmus University Rotterdam
Contents

Part I : A variable metric method for linearly constrained optimization.

Introduction

Chapter I : Variable metric methods for unconstrained optimization.

Section 1 Relations for finding the optimum of quadratic objective functions.

Section 2 Principles of variable metric methods.

Chapter II : Quasi Newton methods with linear constraints.

Section 1 Goldfarb's method.

Section 2 Murtagh and Sargent's algorithm.

Chapter III : Description of the implemented algorithm.

Section 1 Differences with other algorithms.

Section 2 The algorithm.

Part II : A method to solve the general nonlinear optimization problem with nonlinear constraints.

Introduction.

Chapter I : Theoretical backgrounds.

Chapter II : The implemented algorithm.
Part III: Decomposition methods to ensure numerical stability, and derivation of update formulae.

Chapter I: Matrix notations used in Part III

Chapter II: Decomposition methods for matrices.

Section 1 General introduction to the applied decomposition methods.
Section 2 A method for LU-decomposition.
Section 3 Cholesky decomposition.
Section 4 Example for making a LU-decomposition.
Section 5 Example for making a Cholesky decomposition.

Chapter III: The rank one updating formulae.

Section 1 Derivation of the rank one formulae.
Section 2 The relation between the updates for the hessian matrix and the inverse hessian matrix.

Chapter IV: Updating methods for the Cholesky decompositions.

Section 1 An algorithm for applying the rank one corrections to the Cholesky decompositions.
Section 2 Modifying the Cholesky decompositions when a constraint is dropped from the active constraint set.

Acknowledgment
References and Testproblems
References
Testproblems
Appendix A Example for VLICO
Appendix B Example for VANOP
Part I

A variable metric method for linearly constrained optimization.
Introduction 1)

Since we expect the future to show a growing use of the nonlinear optimization technique, we have been looking for new candidate algorithms, which can possibly replace the MAP code. The MAP code was developed around 1960 by Griffith and Stewart [12], and is often used for solving general nonlinear optimization problems. It seeks a local minimum to the general nonlinear programming problem by solving a sequence of linear programming problems. The linear programming problems are generated by linearizing both the nonlinear objective function and the nonlinear constraints, around the current iteration point, while stepsize restrictions are added every iteration. Although the MAP code is a very robust method, it has as a disadvantage its rather slow convergence.

Since in the field of unconstrained optimization the introduction of the quasi-Newton techniques has shown remarkable good results, it seems quite natural to try to extend these techniques to the field of constrained optimization.

As a first step in this direction we have been looking for an algorithm, which implements the idea of the variable metric methods in the constrained optimization problem, where the objective function is nonlinear and the restrictions are linear functions of the problem variables. This report therefore describes a method to solve the linearly constrained optimization problem:

1) The authors are indebted to Mrs. Anke J. Muller-Sloos for editing this report.
Minimize \( f(x) \)

Subject to:
\[
\begin{align*}
    \mathbf{a}_i^T \mathbf{x} &= b_i & i = 1,2,\ldots,m_1 \\
    \mathbf{a}_i^T \mathbf{x} &\leq b_i & i = m_1 + 1,\ldots,m
\end{align*}
\]

Here \( f(x) \) is a convex sufficiently smooth (twice continuously differentiable) function.

In the past methods to solve this type of problems already existed, such as the method of Frank-Wolfe [6], but their performance was rather poor. One of the reasons for this unattractive behaviour is the fact that the underlying methods for solving unconstrained optimization problems of these algorithms are sometimes very inefficient.

Recently new unconstrained optimization algorithms have been developed, and of these especially the variable metric methods have proven to be very valuable. Compared with, for example, the related method of steepest descent, variable metric algorithms show both a faster convergence and an ability to solve unconstrained optimization problems for which the method of steepest descent failed to find a solution.

The basis of the variable metric methods is the fact that in an iteration point \( x_k \), the search direction \( p_k \), is computed using information on the gradient \( \nabla f(x_k) \), and some approximation of the hessian \( \nabla^2 f(x_k) \), the matrix of second order derivatives of the objective function, in that point. Along this direction the minimum of the objective function is determined, giving a new iteration point. From the differences in the gradient of the current and previous iteration point, an updated approximation of the hessian matrix is then calculated, etc....

In the case that \( f(x) \) is a quadratic function quasi Newton (=variable metric) methods converge in at most \( n \) steps, where \( n \) is the dimension of the vector \( x \).

For the case that \( f(x) \) is not quadratic, but possesses the properties given before, convergence has been proven

Introduction.
[12], when started with an approximation which is sufficiently close to the optimum.

In 1969 Goldfarb [10], [11] and Murtagh and Sargent [20] extended the "unconstrained" variable metric method to the case of a nonlinear objective function with linear constraints. Starting from a feasible point, they both used a set of active constraints, restricting the search directions to hyperplanes parallel to those defined by the active constraints, thus generating a sequence of feasible points $x_k$.

The set of active constraints consists of a set of linearly independent constraints, that are binding in the current iteration point. Not all binding constraints have to be in the active constraint set, but only those which we expect to be active at the optimum.

In our code we have implemented a slightly modified version of the algorithm of Murtagh and Sargent. This choice was based on experiments reported by M. Lenard [18] and Himmelblau [13]. To clarify the principles underlying the resulting algorithm we have taken the following approach:

In part I (This part of the report), the optimization method for the nonlinear programming problem with linear constraints is discussed. The contents of this part is:

In Chapter I we discuss some aspects of the quasi Newton methods for unconstrained optimization.

Chapter II deals with the extension of these methods to the case of linear constraints.

In Chapter III the algorithm as implemented in the computer program VLICO (Variable metric method for Linearly Constrained Optimization), is explained.

In part II of this report the extension of the linearly constrained algorithm to the case of nonlinear constraints with its theoretical backgrounds is discussed.

Part I and Part II have both also appeared in the form of a SHELL report [16],[17].

Introduction.
Part III treats the matrix factorizations and the update formulae we have used in the computer program VLICO.

In the appendix a sample problem is given.
Chapter I: Variable metric methods for unconstrained optimization.

Section 1: Relations for finding the optimum of quadratic objective functions.

As every smooth (=twice continuously differentiable) objective function can be approximated by a quadratic function in a neighbourhood of its minimum, variable metric methods have been designed to solve the following unconstrained minimization problem:

\[(1.1) \quad \text{Minimize } q(x) = 0.5x^T A x + b^T x + c,\]

where $A$ is a positive definite symmetric $n \times n$ matrix, $c$ is a known scalar and the constant vector $b$ and the vector of unknowns $x$ belong to $\mathbb{R}^n$. Then the gradient vector is:

\[(1.2) \quad \nabla q(x) = A x + b,\]

and the difference of two gradients of the objective function can be given as:

\[(1.3) \quad \nabla q(x_1) - \nabla q(x_2) = A (x_1 - x_2).\]

A necessary and sufficient condition for $x^*$ to be the minimum of $q(x)$ is:

\[(1.4) \quad \nabla q(x^*) = \varnothing.\]
From (1.3) and (1.4) it now follows that:

\[(1.5) \quad \nabla q(x) = A (x - x^*) \quad \text{or} \quad \text{or} \]

\[(1.6) \quad x^* = x - A^{-1} \cdot \nabla q(x) \]

In the case that A is a positive semidefinite matrix, the matrix \(A^{-1}\) in the expression (1.6) does not exist. In this case however, we can obtain a good approximation of \(x^*\) by adding a perturbation matrix to the matrix A, to obtain \(\hat{A}\), and consequently compute \(\hat{x}^*\), and use this matrix in relation (1.6). According to relation (1.6) the minimum \(x^*\), of a quadratic function can be calculated if in a point \(x\), the gradient \(\nabla^2f(x)\), and the inverse of the hessian matrix \(H\) (or \(\nabla^2f(x)=A\) in this case), are known.

Section 2: Principles of variable metric methods.

Sometimes however, we do not have any knowledge about the elements of the matrix \(A^{-1}\), or our approximation of this matrix is highly inaccurate. In this case the relations we have derived above are not of much help, and a set of mutually related algorithms, called quasi Newton or variable metric methods, based on these relations have been developed to find the minimum of the function.

For the given quadratic function minimization problem, the variable metric methods generate conjugate search directions, and therefore convergence in at most \(n\) steps is guaranteed.

However in general the objective function \(f(x)\), will not be a quadratic function. For this case convergence has also been proven [19], when started with an initial estimate of \(x^*\) which is sufficiently close to the optimum, and when the function \(f(x)\) is a twice continuously differentiable function. In practice the behaviour of the variable metric methods has proven to be rather insensitive for the quality of this initial estimate. The sensitivity for scaling of the objective function or the variables is extensively treated in Dijkshoorn en Van der Hoek [3].

Chapter I
Variable metric methods for unconstrained optimization.
The variable metric methods are based on the following ideas:

i) An approximation $H_k$ to the hessian $H$ and an estimate $x_k$ of the optimum $x^*$ are given.

ii) According to (1.6) the direction $p_k$, in which the optimum will be looked for is calculated by:

$$-H_k^{-1} \nabla f(x_k).$$

Because $H_k$ is only an approximation to the hessian matrix $H$, a line search along the search direction $p_k$ has to be performed, in order to find the exact minimum along this direction. This is necessary because the proof for finite convergence is based on this exact line minimization. If $\alpha_k$ is the value of $\alpha$ minimizing the objective function $f(x)$ along the line $x_k + \alpha p_k$, then we can calculate the next iteration point as:

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

iii) Because in the neighbourhood of the optimum $x^*$, where the objective function can be approximated accurately by a quadratic function, for the real hessian matrix $H$ the relation:

$$H \cdot (x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$$

holds, the approximation to this hessian matrix $H$ is now updated by adding a matrix $C_k$ of lower rank, so that:

$$H_{k+1} = H_k + C_k,$$

$$H_{k+1} \cdot (x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k),$$

for all preceding values of $k$.

Normally for the correction matrix $C_k$ some matrix of rank one or rank two is taken.

Chapter I
Variable metric methods for unconstrained optimization.
Instead of updating the approximation $H_k$, one can of course also work with an approximation $H'_k$ to $H^{-1}$, and update this estimate in accordance with the relation:

\begin{equation}
(1.11) \quad x_{k+1} - x_k = H^{-1} \{ \nabla f(x_{k+1}) - \nabla f(x_k) \},
\end{equation}

to obtain:

\begin{equation}
(1.12) \quad x_{k+1} - x_k = H'^{-1}_k \{ \nabla f(x_{k+1}) - \nabla f(x_k) \}.
\end{equation}

From the many different possibilities to update the matrix $H_k$ or $H'_k$, we have chosen the rank one correction formulae. For an exact formulation and a complete derivation of these formulae see Part III, Chapter III. Reasons for choosing the rank one corrections are:

1) Rank one methods are less sensitive for an inexact line minimization.

2) Less computational work is required.

3) In the case of linearly constrained optimization it leads to simpler recurrence relations.
Chapter II Quasi Newton methods with linear constraints.

Section 1: Goldfarb's method.

In 1969 Goldfarb [3,4] and Murtagh and Sargent [5] extended the principle of quasi Newton methods to the case of the linearly constrained problem:

\[
\text{(2.1) } \begin{align*}
\text{Minimize } f(x) \\
\text{Subject to:} \\
\bar{a}_i^T x = b_i \quad i = 1, \ldots, m_1 \\
\bar{a}_i^T x \leq b_i \quad i = m_1 + 1, \ldots, m
\end{align*}
\]

where \( f(x) \) is a twice continuously differentiable function, \( \bar{a}_i \), \( i = 1, \ldots, m \) are known constant vectors, \( b_i \), \( i = 1, \ldots, m \) are known scalars and \( x \) is the vector of unknowns.

Given a feasible initial point \( x_i \), both methods determine which constraints are active in \( x_i \), and form a matrix \( N_i \) of full rank, whose columns are the \( q_i \) linearly independent normals of these active constraints.

Goldfarb's idea was to start with an approximation \( H_i^{-1} \) of the inverse hessian matrix, that has the property:

\[
\text{(2.2) } N_i^T H_i^{-1} = 0 .
\]
Thus the matrix $H^{-1}$ is not of full rank. It then holds for the first search direction, $p_i$:

$\begin{align*}
(2.3) \quad p_i &= -H^{-1}_i \nabla f(x_i), \\
(2.4) \quad N_i^T \cdot p_i &= -N_i^T \cdot H^{-1}_i \nabla f(x_i) = 0.
\end{align*}$

This implies that the search direction $p_i$ is parallel to the hyperspace spanned by the active constraints. Now for the update of the approximate inverse hessian $H^{-1}_k$ update formulae are chosen in such a way that whenever no changes in the active constraint set occur the following implication holds:

$\begin{align*}
(2.5) \quad N_k^T H^{-1}_k &= 0 \implies N_{k+i}^T H^{-1}_{k+i} = N_k^T H^{-1}_k = 0.
\end{align*}$

When the approximate inverse hessian matrix $H$ is updated according to the principle given in (2.5) it then follows that as long as the active constraint set does not change, the relation:

$\begin{align*}
(2.6) \quad N_k^T p_k &= 0, \text{ where } p_k = -H_k^{-1} \nabla f(x_k),
\end{align*}$

holds for all $k$. The search direction will then stay parallel to the hyperplane defined by the set of active constraints.

A useful initial guess for $H^{-1}_i$, given a set of active constraints and corresponding matrix $N_i$, is according to [7], the projection matrix:

$\begin{align*}
(2.7) \quad H^{-1}_i &= I - N_i (N_i^T N_i)^{-1} N_i^T.
\end{align*}$

Also in the case where a constraint is added to or removed from the active constraint set update formulae for $H^{-1}_k$ are needed to guarantee the parallelity of the search direction $p_k$, with respect to the hyperplanes defined by the active constraints.

For the case where a new constraint with corresponding normal vector $n_T$ is added to $N_k$ this implies that $H^{-1}_{k+i}$ has to be formed in such a way that:

$\begin{align*}
(2.8) \quad H^{-1}_{k+i} n_T &= 0.
\end{align*}$

Chapter II
Quasi newton methods with linear constraints.
This condition ensures that the search direction $d_{k+1}$ is not only parallel to the previous active constraints, but also to the new active constraint.

In the case of deletion of a constraint with normal vector $n_T$,

\[ H^{-1}_{k+1} n_T \neq 0 \]

should hold. This condition, which increases the rank of the matrix $H^{-1}_{k+1}$ with one, is necessary to ensure that the search direction $d_{k+1}$ is no longer parallel to the now inactive constraint.

Section 2 Murtagh and Sargent's algorithm.

Murtagh and Sargent [20] developed a similar algorithm as Goldfarb, but, instead of updating a matrix $H^{-1}_{k}$ of rank $n-q_k$, they suggest to use an approximation of full rank of the Hessian matrix, and to project the search direction on the space spanned by the set of active constraints.

They first compute an unconstrained quasi-Newton search direction:

\[ d_k^* = -H_{k}^{-1} \nabla f(x_k) \]

and then project it, by premultiplying it with a projection matrix $P_k$ in the metric induced by the positive definite matrix $H_k$.

\[ P_k = I - H_k^{-1} N_k (N_k^T H_k^{-1} N_k)^{-1} N_k^T \]

to obtain the constrained search direction, $p_k$. In a normal iteration, when the set of active constraints does not change, any variable metric updating formula can be used for modifying $H_k$. Recurrence relations are also used for updating $(N_k^T H_k N_k)^{-1}$ after a quasi-Newton correction in $H_k$, thus avoiding the need to recalculate $(N_k^T H_k N_k)^{-1}$ in each new iteration. Also for the case where the set of active constraints and its corresponding matrix $N_k$ is modified, special update formulae for $(N_k^T H_k^{-1} N_k)$ are available. The algorithm of Murtagh and Sargent is:

Chapter II
Quasi newton methods with linear constraints.
STEP 1  Take a feasible initial point $x_1$ (One can always find a feasible point using either a phase 1 of an LP-algorithm or some specially designed algorithm.) Determine the set of active constraints in $x_1$, and form the matrix $N_1$, whose columns are the normals of the active constraints. Take $H_1^{-1} = I$ as an initial approximation of the inverse of the hessian matrix. Set the iteration counter $k=1$.

STEP 2  Compute the unconstrained search direction:

\[ p^*_k = -H_k^{-1} \nabla f(x_k) \]

and premultiply $p^*_k$ with the matrix:

\[ P_k = I - H_k^{-1} N_k (N_k^T H_k^{-1} N_k)^{-1} N_k^T \]

to project $p^*_k$ on the space spanned by the set of active constraints to obtain the real search direction:

\[ p_k = P_k p^*_k \]

The set of Lagrange multipliers corresponding to the active constraints, can be calculated during the computation of $p_k$ [15]. The Lagrange multipliers are given by:

\[ \lambda_k = (N_k^T H_k^{-1} N_k)^{-1} N_k^T H_k^{-1} \nabla f(x_k) \]

STEP 3  Check if the constraint with the largest Lagrange multiplier can be dropped from the set of active constraints. If it is dropped the matrix $N_k$ will lose a column and the matrix $(N_k^T H_k^{-1} N_k)^{-1}$ will have to be corrected accordingly. (Murtagh and Sargent describe a method for this correction but because we do not use this method we will not repeat it here.) Set $k=k+1$ after this correction and return to step 2.

Chapter II

Quasi newton methods with linear constraints.
If no constraint can be dropped, perform a test on global convergence:

If $\|p_k\| \leq \epsilon$ Stop.

Otherwise go on to step 4.

**STEP 4** Execute a minimization along the search direction $p_k$, taking into account that the maximum step size $\alpha_k$, which can be taken is equal to the distance from the current iteration point to the nearest constraint along the direction $p_k$:

$$(2.16) \quad \alpha_k = \min_i \left\{ (b_i - a_i^T x_k) / a_i^T p_k \mid a_i^T p_k > 0 \right\},$$

where $a_i$ is the normal of the $i$-th constraint. Set $x_{k+1} = x_k + \alpha_k p_k$ where $\alpha_k$ is the minimizing step size.

**STEP 5** Perform a variable metric correction to $H_k^{-1}$ and $(N_k^T H_k^{-1} N_k)^{-1}$ to obtain $H_{k+1}^{-1}$ and $(N_{k+1}^T H_{k+1}^{-1} N_{k+1})^{-1}$ respectively, where $N_{k+1} = N_k$ as no changes in the active constraint set have occurred.

**STEP 6** If the step size $\alpha_k$, determined in step 4 was equal to the maximum step size, the set of active constraints is increased with the restricting constraint. Consequently a column is added to the matrix $N_k$ to obtain the matrix $N_{k+1}$ and the matrix $(N_{k+1}^T H_{k+1}^{-1} N_{k+1})^{-1}$ is modified accordingly, using the method described by Murtagh and Sargent. We thus obtain $(N_{k+1}^T H_{k+1}^{-1} N_{k+1})^{-1}$.

Set $k = k+1$ and return to step 2.

Chapter II

Quasi newton methods with linear constraints.
Chapter III: Description of the implemented algorithm.

Section 1: Differences with other algorithms.

In our computer code we have used an adapted version of the algorithm of Murtagh and Sargent. The differences are:

1. We only test whether a constraint has to be dropped from the active constraint set in some special situations, where the algorithm of Murtagh and Sargent test every iteration. The situations in which we test are:
   a. The variable metric optimization converged within the current active constraint set.
   b. We have to reinitialize the approximation of the hessian matrix, because of accumulated calculation errors.
   c. After any changes in the set of active constraints, independent whether a constraint is added to it or dropped from it.

2. To obtain an increased numerical stability the approximation $H_k$ to the hessian matrix and the matrix $N_k^T H_k^{-1} N_k$ are used in stead of the matrices $H_k$ and $(N_k^T H_k^{-1} N_k)^{-1}$. $H_k$ and $N_k^T H_k^{-1} N_k$ are stored in the form of their Cholesky decompositions.
By a Cholesky decomposition of a positive definite matrix $C$ is meant the factorization:

$$C = L \cdot D \cdot L^T,$$

where $L$ is a unit lower triangular matrix and $D$ is a diagonal matrix. For more information concerning Cholesky decompositions, and for a method for making a Cholesky decomposition, see Part III Chapter II, Section 3.

Using Cholesky decompositions offers 3 advantages:

a. Calculations can be executed with greater speed.

b. Positive definiteness of $H_k$ and $N_k^T H_k^{-1} N_k$ can easily be controlled.

c. A greater numerical stability is obtained, by ordering the diagonal elements and corresponding rows and columns on their absolute magnitude.

Because update formulae are given [7] for $H_k$ and $N_k^T H_k^{-1} N_k$, and not for their Cholesky factorizations, other modification formulae were needed in this case. Gill, Golub, Murray and Saunders [8, 9] have developed an efficient algorithm to update the Cholesky factors of a matrix, when a matrix of the form $v \cdot v^T$ is added to the original matrix or subtracted from it. We have used this algorithm because the corrections we have to apply to the matrices $H_k$ and $N_k^T H_k^{-1} N_k$ are indeed of the form $v \cdot v^T$. See Part III Chapter IV.

As no suited update formulae could be found in literature to update the Cholesky decompositions of $N_k^T H_k^{-1} N_k$, when changes in $N_k$ occur, special update relations had to be developed for this situation.

Adding a column to the matrix $N_k$ does not give many problems in modifying the matrix $N_k^T H_k^{-1} N_k$, because this amounts to adding a column and a row to the matrix $N_k^T H_k^{-1} N_k$, and we can simply take one more step in the original process of making a Cholesky decomposition, see therefor [21], and Part III Chapter II, Section 3 of this report. Removing column $i$ from the matrix $N_k$ however, amounts to deleting column $i$ and row $i$ in the matrix $N_k^T H_k^{-1} N_k$ to obtain $N_{k+1}^T H_{k+1}^{-1} N_{k+1} = N_{k+1}^T H_{k+1}^{-1} N_{k+1}$.

Chapter III
Description of the implemented algorithm.
No method to calculate the Cholesky factors \( L_{\kappa\kappa} \) and \( D_{\kappa\kappa} \) of the matrix \( \mathbf{N}_{\kappa\kappa} \mathbf{B}_{\kappa\kappa}^{-1} \mathbf{N}_{\kappa\kappa} \) was available. However, an algorithm for this modification is developed in this report. The method used in the program is described extensively in Part III, Chapter IV, Section 2.

Section 2 The algorithm

The proposed algorithm for our linearly constrained minimization now works as follows:

Step 1 Initialize a feasible starting point \( x_1 \), with its gradient \( \nabla f(x_1) \). Take the unit matrix \( I \) as the first approximation \( H_1 \) to the hessian matrix of the objective function \( f(x) \). Determine the set of active constraints and the corresponding matrix \( N \) consisting of the normals of the active constraints. Compute the Cholesky decomposition of \( N_1^T H_1^{-1} N_1 \), and set the iteration counter \( k=1 \).

Step 2 Determine the search direction:

\[
\mathbf{p}_k = -\mathbf{P}_k \mathbf{H}_k^{-1} \nabla f(x_k),
\]

where \( \mathbf{P}_k = I - \mathbf{H}_k^T \mathbf{H}_k \mathbf{N}_k (\mathbf{N}_k^T \mathbf{H}_k^{-1} \mathbf{N}_k)^{-1} \mathbf{N}_k^T \)

and compute the maximum steplength \( \alpha_k \) along \( \mathbf{p}_k \), so that for \( 0 < \alpha < \alpha_k \), \( x_k + \alpha \mathbf{p}_k \) is a feasible point. (For an exact formulation of \( \alpha_k \) see formula (2.16)). Also calculate the approximation of the Lagrange multipliers:

\[
\alpha_k = (\mathbf{N}_k^T \mathbf{H}_k^{-1} \mathbf{N}_k)^{-1} \mathbf{N}_k^T \mathbf{H}_k^{-1} \nabla f(x_k)
\]

If \( \| \mathbf{N}_k^T \mathbf{p}_k \| > \varepsilon \), where \( \varepsilon \) is a small user supplied constant, the search direction is not parallel to the active constraints; go to step 6. If \( \| \mathbf{p}_k \| < \varepsilon \), or if in iteration \( k-1 \) the set of
active constraints was changed, go to step 7. Otherwise go to step 3.

Step 3 Find the steplength $\alpha_k (0 < \alpha_k < 2\alpha_k)$ that minimizes

$$f(x_n + \alpha_k E_k).$$

Step 4 Set $x_{k+1} = x_k + \alpha_k E_k$, and modify the Cholesky decompositions of $H_k$ and $N_k^T H_k^{-1} N_k$ to obtain $H_{k+1}$ and $N_{k+1}^T H_{k+1}^{-1} N_{k+1}$. If $\alpha_k = 2\alpha_k$, go to step 5. Otherwise set $k = k+1$ and return to step 2.

Step 5 A new constraint has become active. Add the normal of the new active constraint to the matrix $N_k$ to obtain the matrix $N_{k+1}$, and modify the Cholesky factors of the matrix $N_k^T H_k^{-1} N_k$ accordingly. Set $k = k+1$ and return to step 2.

Step 6 Reset the approximation of the hessian matrix $H_k$, to the unit matrix $I$, and the matrix $N_k H_k^{-1} N_k$ to $N_k N_k$.

Step 7 Select the largest lagrange multiplier $\lambda(j)$ and calculate $\beta = 5 \lambda(j)/b(j)$, where $b(j)$ is the $j$-th diagonal element of $(N_k^T H_k^{-1} N_k)^{-1}$. $\beta$ can be interpreted as the expected improvement when constraint $j$ is dropped from the set of active constraints.

Stop the procedure if both $\|p_k\| < \varepsilon$ and $\beta < \varepsilon$. If $\|p_k\| \leq \beta$ drop the $j$-th constraint from the set of active constraints. Update the matrix $N_k$ and modify the Cholesky factors of $N_k^T H_k^{-1} N_k$ using the method described in Part III Chapter IV, section 2, to obtain the matrices $N_{k+1}$ and $N_{k+1}^T H_{k+1}^{-1} N_{k+1}$. Set $x_{k+1} = x_k$ and $k = k+1$; Return to step 2. If no change occurred in the set of active constraints, continue the $k$-th iteration with step 3.

Because this algorithm needs a feasible initial point, two phases are needed. In phase 1 the initial point $x_0$, supplied by the user, is modified to obtain a point which is feasible with respect to the equality constraints. Then a penalty function is formulated, which after minimization yields a feasible point to start the second phase with. The construction of this penalty function is as follows:

Chapter III
Description of the implemented algorithm.
Let $\mathcal{V}$ be the set of indices $i$ for which the constraints
\[ a_i^T x \leq b_i \]
are violated. Then we choose as penalty function:
\[ p(x) = \sum_{i \in \mathcal{V}} (a_i^T x - b_i + 1)^2 (1 + \|b_i\|) \]

We now use the above described algorithm to solve a problem with $p(x)$ as objective function and the non violated constraints as restrictions. Therefore some differences in the procedure are needed:

- We check every iteration if any of the violated constraints is satisfied after the last step, and we change $\mathcal{V}$ if any constraints did become feasible.

- Because the set $\mathcal{V}$ can change every iteration, we do not have the same function every iteration, and it does not make sense to update the matrices $H_\kappa$ and $N_\kappa^T B_\kappa N_\kappa$ if no changes in the set of active constraints occur.

If a feasible point to the linear constraints exist, the method described above is guaranteed to find a feasible point.

Note that the penalty function is constructed so that a point in the interior of the feasible region is generated, if it exists. Reason for this construction is the slow convergence of the quadratic penalty function if one wants to generate a point which lies on the edge of the constraints.

As soon as all constraints are satisfied the phase 1 is stopped, and we use the resulting feasible point as an initial point for the second phase.
Part II

A method to solve the general nonlinear optimization problem with nonlinear constraints.
In recent years the field of nonlinear programming has received very much attention. One reason for this may be that through the now widespread use of linear programming the demand is growing for methods that solve nonlinear problems, which give a better representation of reality. Another important reason may be that the knowledge of the theory underlying nonlinear programming has grown, thus providing the ground on which better algorithms can be developed.

This code is developed as a possible improvement of the MAP code of Griffith and Stewart [12] to solve the following nonlinear programming problem:

\[(1.1) \text{ Minimize } f(x)\]
\[
\text{Subject to:} \\
\quad h_j(x) = 0 \quad j=1,\ldots,m_A \\
\quad g_i(x) \leq 0 \quad i=1,\ldots,m_B
\]

where \( f(x) \), \( h_j(x) \) and \( g_i(x) \) are nonlinear or linear, twice continuously differentiable functions of \( x \in \mathbb{R}^n \).

The MAP code transforms this problem to a sequence of linear programming problems, generated by linearization of both the nonlinear objective function \( f(x) \) and the nonlinear constraints \( h_j(x) \), \( j=1,\ldots,m_A \) and \( g_i(x) \), \( i=1,\ldots,m_B \), in a neighbourhood of the current iteration point. The
solutions of these LP-problems converge under certain conditions to the solution of (1.1).

Because in these linearizations all information about the nonlinearity of objective function and constraints is discarded, this method has as a disadvantage its slow convergence.

The method we propose in this report may be a possible replacement for the MAP code. This method, based on a report of J.B. Rosen [22], also generates a sequence of problems, the solutions of which will converge to a local minimum of (1.1). The problems generated in the latter method are created by linearization of the nonlinear constraints only, whereas the original objective function is replaced by a modified Lagrangian function:

\[
m(x) = f(x) + \sum_{j \in J} \lambda_j [ h_j(x) - l_j(x,y) ] + \sum_{i \in I} \mu_i [ g_i(x) - l_i(x,y) ]
\]

Where \( \lambda_j \) and \( \mu_i \) are the Lagrange multipliers of the nonlinear constraints \( h_j \) and \( g_i \), and \( l_j(x,y) \) and \( k_i(x,y) \) are the linearizations of these constraints with respect to the point \( y \).

Contrary to MAP this method thus preserves information regarding the nonlinearity of both the objective function, and the active nonlinear constraints, in this way obtaining a faster and robust convergence. Where MAP generates a sequence of LP problems, the proposed method creates a series of problems of the form:

\[
(1.2) \text{ minimize } m(x)
\]

Subject to :

\[
\mathbf{a}_j^T x = b_j \quad j=1, \ldots, m_1
\]

\[
\mathbf{a}_i^T x \leq b_i \quad i=m_1+1, \ldots, m_2
\]

Where \( m(x) \) is a nonlinear function.

Introduction.
The method we have used for solving problem (1.2) is the program VLICO, which is discussed in Part I of this report, but the program VLICO can be replaced by any other method that solves problem (1.2).

Chapter I of this part will give some theoretical backgrounds of the algorithm, whereas Chapter II treats the implementation of the algorithm.
Chapter I Theoretical backgrounds.

If we make a distinction between linear and nonlinear constraints in the general nonlinear optimization problem, we can define the regions $S_1$ and $S_2$ as follows:

(1.1) $S_1 = \left\{ x \mid \begin{array}{l} \mathbf{a}_i^T \mathbf{x} - b_i = 0 \ ; \ i = 1, \ldots, m_1 \\ \mathbf{a}_i^T \mathbf{x} - b_i \leq 0 \ ; \ i = m_1 + 1, \ldots, m_2 \end{array} \right\}$

$S_1$ is the feasible region defined by the linear constraints only, and:

(1.2) $S_2 = \left\{ x \mid \begin{array}{l} h_j(x) = 0 \ ; j = 1, \ldots, m_3 \\ g_i(x) \leq 0 \ ; i = 1, \ldots, m_4 \end{array} \right\}$

$S_2$ is the feasible region defined by the nonlinear constraints only.

We can write the problem we want to solve now as:

(1.3) Minimize $f(x)$

subject to:

$x \in S_1 \cap S_2$

Rosen's method uses the linear approximations to the nonlinear constraints, and generates the following sequence of problems, the solutions $\{x_k\}$ of which converge to a solution of problem 1.3.

Chapter I Theoretical backgrounds.
(1.4) Minimize \( m(x) = f(x) + s(x, x_{k-1}) \)
subject to:
\[ x \in S_1 \cap T_2(x_{k-1}) \]
where \( T_2(x_{k-1}) \) is the region created by linearization of the nonlinear constraints:
\[
(1.5) \quad T_2(x_{k-1}) = \left\{ x \mid \begin{array}{ll}
  l_j(x, x_{k-1}) = 0 & ; j = 1, \ldots, m_3 \\
  k_i(x, x_{k-1}) = 0 & ; i = 1, \ldots, m_4
\end{array} \right\}
\]
where the linearization of \( h_i(x) \), \( l_j(x, x_{k-1}) \), is defined as:
\[
(1.6) \quad l_j(x, x_{k-1}) = h(x) + (x - x_{k-1})^T \nabla h_j(x_{k-1})
\]
and \( k_i(x, x_{k-1}) \) is defined in the same way as linearization of \( g_k(x) \). Rosen [22] has proved that if a solution \( x_k \) to (1.4) is a fixed point of the mapping of \( E^m \rightarrow E^m \), defined by the algorithm then \( x_k \) is also a Kuhn-Tucker point of problem (1.3). The convergence of the series \( \{x_k\} \) to a solution of (1.3) of course depends heavily on the function \( s(x, x_{k-1}) \). Further we want the function \( s(x, x_{k-1}) \) to possess the following properties:
\[
(1.7) \quad s(x_{k-1}, x_{k-1}) = 0,
(1.8) \quad \nabla s(x_{k-1}, x_{k-1}) = 0.
\]
Because when \( x_k = x_{k-1} \), i.e. the solution to (1.3) is found, we want the following relations to hold:
\[
(1.9) \quad m(x_k) = f(x_k), \quad \text{and}
(1.10) \quad \nabla m(x_k) = \nabla f(x_k).
\]
The modified Lagrange function is an objective function \( m(x) \) that possesses properties (1.9) and (1.10), and of which fast convergence is proven for the sequence \( \{x_n\} \).
But because we do not know the exact values of the Lagrange multipliers we have to use the approximations available. Using the modified Lagrange function as objective function corresponds with taking:

Chapter I
Theoretical backgrounds.
\[ (1.11) \quad s(x,x_{k-1}) = \sum_{i=1}^{m_h} \lambda_i(x_{k-1}) \{h_i(x) - 1_i(x_{k-1},x)\} + \sum_{j=1}^{m_u} \mu_j(x_{k-1}) \{g_j(x) - k_j(x,x_{k-1})\} \]

where \( \lambda_i(x_{k-1}) \) and \( \mu_j(x_{k-1}) \) are approximations to the Lagrange multipliers of the constraints \( h_i(x) \) and \( g_j(x) \) in the point \( x_{k-1} \). Note that the function \( s(x,x_{k-1}) \) contains information concerning the nonlinearity of the constraints. Convergence proofs for this objective function exist, when the starting point \( x_i \) is sufficiently close to a local minimum of (1.3). Starting from an arbitrary point \( x_i \), however, no convergence can be guaranteed.

To overcome this difficulty, we introduce a first phase to obtain a good starting point \( x_i \), with corresponding estimates of the Lagrange multipliers \( \lambda_i(x_i) \) and \( \mu_j(x_i) \). The method we have used for this first phase, is solving the problem:

\[ (1.12) \quad \text{Minimize} \quad f(x) + p(x) \]

subject to:
\[ x \in \mathcal{S}_1, \]

where \( p(x) \) is defined as:

\[ (1.13) \quad p(x) = 0.5 \cdot \Pi_i \left( \sum_{j=1}^{m_h} \{h_j(x)\}^2 + \sum_{i=1}^{m_u} \{g_i^+(x)\}^2 \right), \]

where \( \Pi_i \) is a penalty parameter, and \( g_i^+(x) \) is defined as:

\[ (1.14) \quad g_i^+(x) = \max \{0, g_i(x)\} \]

The objective function of problem (1.12) is the external penalty function of the SUMT procedure of Fiacco and McCormick [5], but, where the SUMT methods generate a sequence of points \( \{x_k\} \) that converges to a local minimum, by increasing the value of the penalty parameter \( \Pi_i \) every step, we take only one SUMT step. After this SUMT step, we solve a series of the form (1.4). Besides providing a good initial estimate \( x_i \), the solution to problem (1.12) also gives good approximations of the Lagrange multipliers \( \lambda_i(x_i) \) and \( \mu_j(x_i) \) in \( x_i \), given as:

Chapter I
Theoretical backgrounds.
(1.15) \[ \lambda_i(x_i) = \pi_i \cdot h_i(x_i) \]

(1.16) \[ \mu_j(x_j) = \pi_i \cdot g_j^r(x_j) \]

It can be shown [22], that if \( \pi_i \) is chosen greater than some constant, the sequence \( \{x_k\} \) will converge to a local minimum of (1.3). Therefore the algorithm consists of a SUMT step followed by a sequence of linearized problems.
Chapter II: The Implemented Algorithm.

The algorithm as we have implemented it in our computer code is:

The input data are:

- Initial approximation $x_0$; the linear constraints $h_i(x)$ and $g_j(x)$;
- Parameters for VLICO and VAMP, such as precision parameters and penalty parameters.

**Step 1** Solve with VLICO the problem:

\[
\min f(x) + p(x), \quad x \in S_1,
\]

subject to $x \in S_1$, where $S_1$ is defined in (1.1) and $p(x)$ in (1.13), starting in point $x_0$ to obtain $x_1$ and $\lambda_1$. Set $x_k = x_1$, $\lambda_k = \lambda_1$, and $k = 1$. If no point $x \in S_1$ can be found, stop because it is an infeasible problem.

**Step 2** Given $x_k$ and $\lambda_k$ generate the feasible region $T_2(x_k)$ as defined in (1.5).

Chapter II
The implemented algorithm.
Step 3 Solve with VLICO the problem:

\[(2.2) \text{ Minimize } f(x) + s(x, x_k) \]
subject to \( x \in T_2(x_k) \cap S_1 \)

where \( s(x, x_k) \) is given in (1.11), to obtain the vectors \( x_{k+1} \) and \( \lambda_{k+1} \).

Step 4 If \( \|x_{k+1} - x_k\| \leq \epsilon \), where \( \epsilon \) is some predetermined constant, stop because the method has converged.

Step 5 Set \( x_{k+1} = x_k \); \( \lambda_{k+1} = \lambda_k \) and \( k = k+1 \), and return to step 2.

It should be noted that the principle of the algorithm is independent of the VLICO code. The VLICO program can be replaced by any other computer program which solves the linearly constrained problem.

Possible extensions which can easily be incorporated in this algorithm are:

- Termination in case of infeasibility caused by the nonlinear constraints.
- Early recognition of optimal initial points.
- Termination in case of exceeding a predetermined maximum number of iteration steps.
- Linearization of only a subset of the nonlinear constraints, instead of linearizing all nonlinear constraints.

Chapter II
The implemented algorithm.
Part III

Decomposition methods to ensure numerical stability, and derivation of update formulae.
Chapter I  Matrix notations used in part III

1. A lower trapezoidal matrix $L$ is a $m \times n$ ($m \geq n$) matrix $l(i,j)$, for which the following relation holds:

\[
l(i,j) = 0 \text{ for } j = i+1 \text{ to } n, \text{ and } i = 1 \text{ to } n.
\]

In a picture:

![Figure 1](image)

Figure 1
2. A lower triangular matrix $L$ is a $n \times n$ matrix $l(i,j)$ for which the following relation holds:

$$l(i,j) = 0 \text{ for } j=i+1 \text{ to } n, \text{ and } i=1 \text{ to } n.$$ 

In a picture:

![figure 2]

3. An upper triangular matrix is a transposed lower triangular matrix.

4. A unit lower (or upper) triangular matrix is a lower (or upper) triangular matrix with all diagonal elements equal to 1.

Chapter I

Matrix notations.
5. A 'special' triangular matrix $M(p, b, g)$ is a triangular matrix $m(i, j)$ for which the following relations hold:

$$m(i, j) = 0 \text{ for } j = i + 1 \text{ to } n, \text{ and } i = 1 \text{ to } n$$

$$m(i, j) = c(i) \text{ for } j = i, \text{ and } i = 1 \text{ to } n.$$ 

$$m(i, j) = p(i)b(j) \text{ for } j = 1 \text{ to } i - 1, \text{ and } i = 1 \text{ to } n.$$ 

6. An elementary matrix $E_k$ is a $n \times n$ triangular matrix $e(i, j)$, for which the following relations hold:

$$e(i, j) = 1 \text{ for } j = i, \text{ and } i = 1 \text{ to } n$$

$$e(i, j) = 0 \text{ for } j = i + 1 \text{ to } n, \text{ and } i = 1 \text{ to } n$$

$$e(i, j) = 0 \text{ for } i = j + 1 \text{ to } n, \text{ and } j = 1 \text{ to } k - 1$$

$$e(i, j) = 0 \text{ for } i = j + 1 \text{ to } n, \text{ and } j = k + 1 \text{ to } n$$

In a picture:

![Figure 3](image-url)
7. A diagonal matrix $D$ is a $n \times n$ square matrix $d(i,j)$, for which the following relation holds:

$$d(i,j) = 0 \text{ for } j \neq i, \text{ and } i, j = 1 \text{ to } n$$

In a picture:

![Diagram of a diagonal matrix]

**Figure 4**

8. The unit matrix $I$, is a diagonal matrix with all diagonal elements equal to $1$. 

---

Chapter I
Matrix notations.
9. A permutation matrix $P$ is a $n \times n$ matrix $p(i,j)$, for which the following relations hold:

$$\sum_{i=1}^{n} p(i,j) = 1 \quad j = 1 \text{ to } n$$
$$\sum_{j=1}^{n} p(i,j) = 1 \quad i = 1 \text{ to } n$$

$p(i,j) = 0$ or $p(i,j) = 1$.

Picture for an example with $n = 4$:

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 0 1
```

Figure 5

Chapter I
Matrix notations.
Chapter II Decomposition methods for matrices.

Section 1 General introduction to the applied decomposition methods.

By a decomposition of a certain matrix we understand a procedure that forms a number of matrices (usually two or three) with some special features, e.g. triangularity, such that the product of these matrices yields the original matrix.

In our optimization method, we have made use of LU-decomposition and Cholesky decomposition. The LU-decomposition of a $m \times n$ ($m \geq n$) matrix $A$ consists of a $m \times n$ lower trapezoidal matrix $L$ and an $n \times n$ unit upper triangular matrix $U$, such that:

\[(2.1.1) \quad A = LU\]

The elements of a Cholesky decomposition of a positive definite symmetric matrix $B$, are a $n \times n$ unit lower triangular matrix $L$, and a diagonal matrix $D$, such that:

\[(2.1.2) \quad B = LDL^T\]

The advantage of the decomposed form of a matrix is that many operations with matrices can be executed in a simpler way with a greater speed and accuracy when a matrix is of some special form.
To compute the inverse of a nonsingular \( n \times n \) matrix \( A \), for example, is very simple if the LU-factorization (2.1.1) of this matrix is known. The matrix \( A' \) is then equal to \( A' = U' \cdot L' \) and the matrices \( U' \) and \( L' \) can be computed by simple recurrence formulae.

In our research we have used LU- and Cholesky decompositions to obtain numerically stable solutions of the set of equations:

\[
(2.1.3) \quad A \cdot x = b
\]

If \( A \) is a positive definite symmetric \( n \times n \) matrix, and we have its Cholesky decomposition, then this set of equations is equivalent to:

\[
(2.1.4) \quad L D L^T \cdot x = b
\]

We can solve \( x \) from (2.1.4) by solving the following sequence:

\[
(2.1.5) \quad \text{solve } y \text{ from } L y = b
\]

\[
(2.1.6) \quad \text{compute } w = D^{-1} y
\]

\[
(2.1.7) \quad \text{solve } x \text{ from } L^T x = w
\]

Here the solutions of (2.1.5) and (2.1.7) can easily be obtained by means of a simple backsubstitution thanks to the lower and upper triangular structure of \( L \) and \( L^T \). To compute \( w \) in (2.1.6) also forms no problem because \( D \) is a diagonal matrix.

In the case that \( A \) in (2.1.3) is not a square matrix but a \( m \times n \) matrix \((m>n)\), we make a LU-factorization of \( A \) and solve only the first \( n \) variables of \( x \) and leave the rest of the variables unchanged.

In the sections 2 and 3 of this chapter we will treat methods for making a LU-factorization and a Cholesky decomposition. In sections 4 and 5 examples of the decomposition methods are given.
Section 2 A method for LU-decomposition.

This method [21] for finding a m\times n lower trapezoidal matrix L and a n\times n unit upper triangular matrix U, such that: \( A=LU \), generates a sequence \( L_0U_0, \ L_1U_1, \ L_2U_2, \ldots, \ L_nU_n \), where \( L=L_n \) and \( U=U_n \), and the relation \( A=L_0U_0 \) always holds; for a practical example see section 4 of this chapter.

Start with \( L_0=A \) and \( U_0=I \), and form the elementary matrix \( E_1 \) such that for the matrix \( L_1=L_0E_1 \) it holds that the elements \( L_1(1,j) \) for \( j>1 \) are zero. Now compute the matrix \( E_1^{-1} \) (This is also an elementary matrix \( E_1^{-1} \)) and premultiply \( U_0 \) with this matrix to obtain \( U_1=E_1^{-1}U_0 \).

Now \( L_1U_1=L_0E_1E_1^{-1}U_0=A \cdot I \cdot I=A \). After this we form in the same way matrices:

\[
(2.2.1) \quad L_k=L_0E_1E_2\ldotsmE_k, \text{ and }
\]

\[
(2.2.2) \quad U_k=E_k^{-1}E_{k-1}^{-1}\ldotsmE_1^{-1}U_0
\]

Where every time the matrices \( E_1 \) to \( E_k \), and \( E_k^{-1} \) to \( E_1^{-1} \) are chosen in such a way that the relation:

\[ L_k(i,j)=0 \text{ for } j>i, \text{ and } 1\leq k, \]

holds for the matrix \( L_k \), and \( E_1E_1^{-1}=I \) for \( i=1 \) to \( k \). Now for the matrix \( U_k \) it then holds that:

\[ U_k(i,i)=1 \text{, and } \]

\[ U_k(i,j)\neq 0 \text{ only for } i<j \text{ and } 1\leq k, \]

and all other elements are equal to zero. Proceeding in this way we obtain \( L=L_n \) and \( U=U_n \).

For the stability we do not make a LU-decomposition of the matrix \( A \) itself, but of the matrix \( PA \), where \( P \) is a permutation matrix, which can be written as:

\[
(2.2.3) \quad P=P_1P_2\ldotsmP_n
\]

where each matrix \( P_k \) is a matrix that interchanges two rows in a matrix. Now each matrix \( P_k \) is chosen in such a way that in the matrix
the absolutely largest element in the \( k \)-th column is placed on the \( k \)-th diagonal place.

The total number of required multiplications for making a LU-decomposition is:

\[
(2.2.4) \quad \frac{5}{2} m n^2 - \frac{n^3}{6} + O(n^2)
\]

Section 3 Cholesky decomposition.

The method we have used for finding a unit lower triangular matrix \( L \) and a diagonal matrix \( D \) such that the symmetric positive definite matrix \( A \) can be written as:

\[
(2.3.1) \quad A = LDL^T,
\]

is a recursive method. For a practical example of this method see section 5 of this chapter.

Suppose that a \( n \times n \) symmetric positive definite matrix \( A_n \) can be written as:

\[
(3.3.2) \quad A_n = \begin{bmatrix}
A_{n-1} & b \\
b^T & a_{nn}
\end{bmatrix}
\]

where we know the Cholesky decomposition of the \((n-1)\times(n-1)\) matrix \( A_{n-1} \):

\[
(3.3.3) \quad A_{n-1} = L_{n-1}.D_{n-1}.L_{n-1}^T
\]

Then we can find the Cholesky-factors \( L_n \) and \( D_n \) of the matrix \( A_n \) in the following way:

If we put:

\[
L_n = \begin{bmatrix}
L_{n-1} & 0 \\
0 & c
\end{bmatrix}, \quad \text{and} \quad D_n = \begin{bmatrix}
D_{n-1} & 0 \\
0 & x
\end{bmatrix}
\]
Then we derive the following relation:

\[
(3.3.4) \begin{bmatrix} A_{n-1} & b \\ b^T & a_{nn} \end{bmatrix} \begin{bmatrix} b_{n-1} \\ a_{n-1} \end{bmatrix} = A_n = L_n \cdot D_n \cdot L_n^T = \begin{bmatrix} L_{n-1} \cdot D_{n-1} \cdot L_{n-1}^T & L_{n-1} \cdot D_{n-1} \cdot \gamma \\ \zeta' \cdot D_{n-1} \cdot L_{n-1}^T & \zeta' \cdot D_{n-1} \cdot \gamma + x \end{bmatrix}
\]

Now we can compute \( \gamma \) from the relation

\[
(3.3.5) \quad L_{n-1} \cdot D_{n-1} \cdot \gamma = b
\]

by a backsubstitution because \( L_{n-1} \) is a triangular matrix. We can compute \( x \) from:

\[
(3.3.6) \quad \zeta' \cdot D_{n-1} \cdot \gamma + x = a_{nn}.
\]

The relation between the positive definiteness of \( A_n \) and the sign of \( x \) can easily be derived:

If \( A_n \) is positive definite, then so is \( A_{n-1} \), and:

\[
\zeta \delta(\det(A_n)) = \det(L_n) \cdot \det(D_n) \cdot \det(L_n^T) = \det(D_{n-1}) \cdot x,
\]

so \( x \) is positive because both \( \det(D_n) \) and \( \det(D_{n-1}) \) are positive. Now the Cholesky factors can easily be computed using the above derived recurrence relations, because \( L_1 = 1 \), and \( D_1 = A(1,1) \).

To ensure stability it is advisable to order the diagonal elements of the matrix \( A \) on their magnitude, i.e. form the matrix \( P \cdot A \cdot P' \), where \( P \) is a permutation matrix. The total number of multiplications required to make a Cholesky decomposition of a matrix of order \( n \) is:

\[
(3.3.7) \quad n^3 / 3 + \mathcal{O}(n^2)
\]

Chapter II
Decomposition methods for matrices.
Section 4 Example for making a LU-decomposition.

Suppose we want to make a LU-decomposition of the matrix:

\[
A = \begin{bmatrix}
1 & 2 & 0 \\
5 & 3 & 2 \\
4 & 1 & 3 \\
2 & 1 & 5 \\
0 & 2 & 2
\end{bmatrix}
\]

Then we will have to start with the following matrices \( L_0 \) and \( U_0 \):

\[
L_0 = A, \quad \text{and} \quad U_0 = I.
\]

Now for the first step we look for the largest element in the first column, and we interchange the rows 1 and 2. After this we postmultiply \( L_0 \) by the elementary matrix \( E_1 \), and premultiply \( U_0 \) by \( E_1^{-1} \), where \( E_1 \) and \( E_1^{-1} \) are:

\[
E_1 = \begin{bmatrix}
1 & -3/5 & -2/5 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}, \quad \text{and} \quad E_1^{-1} = \begin{bmatrix}
1 & 3/5 & 2/5 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

And we obtain: \( U_1 = E_1^{-1} \), and:

\[
L_1 = \begin{bmatrix}
5 & 0 & 0 \\
1 & 7/5 & -2/5 \\
4 & -7/5 & 7/5 \\
2 & -1/5 & 21/5 \\
0 & 2 & 2
\end{bmatrix}
\]

For the second step we first select the absolutely largest element in column 2, and accordingly interchange row 2 and row 5. Now we postmultiply \( L_1 \) by the matrix \( E_2 \), and premultiply \( U_1 \) by \( E_2^{-1} \), where \( E_2 \) and \( E_2^{-1} \) are given by:

\[
E_2 = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & -1 \\
0 & 0 & 1
\end{bmatrix}; \quad E_2^{-1} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 0 & 1
\end{bmatrix}
\]

After this step we obtain:
Section 5  Example for making a Cholesky decomposition

Suppose we want to make a Cholesky decomposition of the positive definite symmetric matrix $A$:

\[
(2.5.1) \quad A = \begin{bmatrix}
124/75 & 1 & 13/10 \\
1 & 3 & 3/2 \\
13/10 & 3/2 & 11/4
\end{bmatrix}
\]

To ensure stability, we will first have to order the matrix on its diagonal elements, and we obtain:

\[
(2.5.2) \quad E = \begin{bmatrix}
3 & 3/2 & 1 \\
3/2 & 11/4 & 13/10 \\
1 & 13/10 & 124/75
\end{bmatrix}
\]

Now we can start making the Cholesky decomposition. We first take $l_1=1$ and $d_1=E(1,1)=3$. We now know that:
Part III

(2.5.3) \( L^2 = \begin{bmatrix} 1 & 0 \\ c_1 & 1 \end{bmatrix}, \) and \( D^2 = \begin{bmatrix} 3 & 0 \\ 0 & x_2 \end{bmatrix} \)

And we can solve \( c_1 \) from the equation:

(2.5.4) \( 3c_1 = 3/2 \Rightarrow c_1 = 1/2 \)

and \( x_2 \) from:

(2.5.5) \( 3/4 + x_2 = 11/4 \Rightarrow x_2 = 2 \)

Our third step consists of computing \( L \) and \( D \) as:

(2.5.6) \( L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ c_1 & c_2 & 1 \end{bmatrix}, \) and \( D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & x_3 \end{bmatrix} \)

If we take \( \mathbf{c} = (c_1, c_2)^T \), we can compute \( \mathbf{c} \) from \( L^2 D^2 \mathbf{c} = \begin{bmatrix} 1 \\ 13/10 \end{bmatrix}^T \), this corresponds with:

(2.5.7) \( 3c_1 = 1 \), and

(2.5.8) \( 3c_1/2 + 2c_2 = 13/10. \)

and solution of these two equations gives \( c_1=1/3, \) and \( c_2=2/5. \) We can now calculate \( x_3 \) from:

(2.5.8) \( \mathbf{c}^T D^2 \mathbf{c} + x_3 = 124/75 \Rightarrow x_3 = 1 \)

We now have the matrices \( L \) and \( D \) such that the product \( LDL^T = B. \)

Chapter II
Decomposition methods for matrices.
Chapter III  The rank one updating formulae.

Section 1  Derivation of the rank one formulae.

Suppose we denote the $k$-th approximation of the hessian matrix by $A_k$, and we want to derive the simplest correction matrix $C_k$ such that the matrix:

$$A_{k+1} = A_k + C_k,$$

satisfies the equation:

$$A_{k+1} (x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k).$$

This equation springs from the requirement that, if the object function were a quadratic function, the matrix $A_{k+1}$ must have one of the properties of the real Hessian matrix, that the matrix $A_k$ did not possess. Then if we use the following relations:

$$x_k = x_{k+1} - x_k,$$

$$y_k = \nabla f(x_{k+1}) - \nabla f(x_k),$$

$$v_k = y_k - A_k z_k$$

inserting (3.1.1) in (3.1.2) yields:
(3.1.6) \( C_k z_k = y_k \).

This equation does not determine \( C_k \) uniquely since it contains only \( n \) conditions for the \( n(n + 1)/2 \) unknown variables of the symmetric \( n \times n \) matrix \( C_k \).

The simplest possible matrix \( C_k \) that fulfills condition (3.1.6) is a matrix of the form:

(3.1.7) \[ C_k = r_k w_k w_k^\top, \]

where \( w_k \) is an \( n \)-dimensional vector, and \( r_k \) is some constant. The correction of this form is called the rank one modification formula, and the scalar \( r_k \) and the vector \( w_k \) are uniquely determined.

Inserting (3.1.7) in (3.1.6) yields:

(3.1.8) \[ r_k w_k w_k^\top z_k = y_k, \]
(3.1.9) \[ w_k = q_k v_k, \]

where \( q_k = 1/(r_k w_k^\top z_k) \) is some unknown scalar. Using (3.1.9) on its turn in relation (3.1.8) yields:

(3.1.10) \[ r_k q_k^{-1} v_k w_k^\top z_k = y_k \]

from which expression we can derive:

(3.1.11) \[ r_k q_k^{-1} = 1/(y_k^\top z_k) \]

Substituting (3.1.9) in (3.1.7) and using (3.1.11) now gives the desired formula:

(3.1.12) \[ C_k = r_k q_k^2 v_k w_k^\top = y_k v_k^\top/(y_k^\top z_k) \]

This suffices if we want to update the approximation of the Hessian matrix. On the other hand if we want to update an approximation of the inverse of the hessian matrix, we can easily derive a related modification formula.

Let the \( k \)-th approximation of the inverse of the Hessian be the matrix \( B_k \) and let the \( k+1 \)-th approximation be given as:
(3.1.13) \( B_{k+1} = B_k + D_k \).

Then we can write the equivalence of equation (3.1.2) as:

\[(3.1.14) \quad z_k = B_{k+1} y_k \, , \]

If we use the notation:

\[(3.1.15) \quad s_k = z_k - B_k y_k \, , \]

and insert (3.1.13) in (3.1.14) we can write:

\[(3.1.16) \quad D_k y_k = s_k \, . \]

For the rank one update formula we can write \( D_k \) as:

\[(3.1.17) \quad D_k = a_k t_k t_k' \, , \]

where \( a_k \) is a scalar and \( t_k \) a \( n \)-dimensional vector.

Substituting (3.1.17) in (3.1.16) gives:

\[(3.1.18) \quad a_k t_k t_k' y_k = s_k \, , \text{ or} \]

\[(3.1.19) \quad t_k = e_k s_k \, , \]

where \( e_k = 1/(a_k t_k y_k) \) is some unknown scalar. Substituting (3.1.19) in (3.1.18) gives:

\[(3.1.20) \quad a_k e_k s_k y_k = s_k \, , \]

and we can conclude that:

\[(3.1.21) \quad a_k e_k^t = 1/(s_k y_k) \, . \]

Inserting (3.1.19) into (3.1.17), and using (3.1.21) now gives us the desired expression for \( D_k \):

\[(3.1.22) \quad D_k = a_k e_k^t s_k y_k = s_k s_k'/(s_k y_k) \, . \]

Chapter III
The rank one updating formulae.
Section 2 The relation between the updates for the Hessian matrix and the inverse Hessian matrix.

In this section we will prove that, if we have approximations $A_k$ and $E_k$ for the Hessian matrix and the inverse of the Hessian matrix respectively, such that $B_k A_k = I$, this relation will also hold for the matrices $A_{k+1}$ and $E_{k+1}$ if the rank one correction formulae, which have been derived in the first section of this chapter, are used.

Suppose we have:

\begin{align*}
(3.2.1) & \quad B_k A_k = I \\
(3.2.2) & \quad B_{k+1} = B_k + S_k S_k'/(S_k' Y_k) \quad \text{and} \\
(3.2.3) & \quad A_{k+1} = A_k + Y_k Y_k'/(Y_k' Z_k) \quad \text{, then we can write } B_{k+1} A_{k+1} \text{ as:} \\
(3.2.4) & \quad B_{k+1} A_{k+1} = B_k A_k + S_k S_k' A_k/(S_k' Y_k) + \\
& \quad B_k Y_k Y_k'/(Y_k' Z_k) + \\
& \quad S_k S_k' Y_k Y_k'/(S_k' Y_k Y_k' Z_k)
\end{align*}

According to relations (3.1.5), (3.1.15) and (3.2.1) we can write:

\begin{align*}
(3.2.5) & \quad S_k' A_k = (Z_k - Y_k' B_k) A_k = Z_k' A_k - Y_k' A_k - Y_k' \quad \text{, and} \\
(3.2.6) & \quad B_k v = B_k (Y_k - A \cdot Z_k) = B_k Y_k - Z_k = -S_k
\end{align*}

when we apply (3.2.1), (3.2.5) and (3.2.6) to (3.2.4) we obtain:

\begin{align*}
(3.2.7) & \quad B_{k+1} A_{k+1} = I + S_k Z_k \{ S_k' Y_k/(S_k' Y_k Y_k' Z_k) - \\
& \quad \quad - 1/(S_k' Y_k) - 1/(Y_k' Z_k) \}
\end{align*}

The term {.....} can also be written as:

\begin{align*}
(3.2.8) & \quad \{ (S_k' Y_k - S_k' Y_k - Y_k' Z_k)/(S_k' Y_k Y_k' Z_k) \}
\end{align*}
Using equations (3.1.5) and (3.1.15) gives the relations:

\[ (3.2.9) \quad \mathbf{s}_k' \mathbf{y}_k = (\mathbf{z}_k' - \mathbf{Bk} \mathbf{y}_k)' (\mathbf{y}_k' - \mathbf{A}_k \mathbf{z}_k) = \]
\[ = 2 \mathbf{z}_k' \mathbf{z}_k - \mathbf{y}_k' \mathbf{Bk} \mathbf{y}_k - \mathbf{z}_k' \mathbf{A}_k \mathbf{z}_k, \]

\[ (3.2.10) \quad \mathbf{s}_k' \mathbf{y}_k = (\mathbf{z}_k' - \mathbf{Bk} \mathbf{y}_k)' \mathbf{y}_k = \mathbf{z}_k' \mathbf{y}_k - \mathbf{y}_k' \mathbf{Bk} \mathbf{y}_k, \]

\[ (3.2.11) \quad \mathbf{v}_k' \mathbf{z}_k = (\mathbf{y}_k' - \mathbf{A}_k \mathbf{z}_k)' \mathbf{z}_k = \mathbf{z}_k' \mathbf{y}_k - \mathbf{z}_k' \mathbf{A}_k \mathbf{z}_k. \]

Inserting these relations in (3.2.8) reduces the term between brackets to zero, and from this fact follows that \( \mathbf{B}_{k+1} \mathbf{A}_{k+1} = \mathbf{I} \), and we have completed the proof.

Chapter III
The rank one updating formulae.
Chapter IV Updating methods for the Cholesky decomposition.

Section 1 An algorithm for applying the rank one corrections to the Cholesky decompositions.

In the optimization method for linearly constrained problems which we have used in our program, an approximation $A_k$ of the Hessian matrix of the objective function is updated every iteration by adding some matrix of rank one to it. This correction matrix has the form:

$$C_k = r_k w_k w_k'$$

where $r_k$ is a scalar and $w_k$ is a $n$-dimensional vector. As in our program we store the matrix $A_k$ in the form of its Cholesky decomposition:

$$A_k = L_k D_k L_k'$$

where $L_k$ is a unit lower triangular matrix, and $D_k$ is a diagonal matrix, we need a set of modification formulae for the Cholesky factors $L_k$ and $D_k$ which correspond with the correction of the matrix $A_k$:

$$A_{k+1} = A_k + r_k w_k w_k'$$
In [8] Gill, Murray and Saunders show simple recurrence relations which yield the Cholesky factors $L_{k+1}$ and $D_{k+1}$ of the matrix:

$$(4.1.4) \quad A_{k+1} = L_{k+1}D_{k+1}L_{k+1}'$$
given the equations (4.1.3) and (4.1.2) where the matrices $L_k$ and $D_k$, the vector $w_k$, and the scalar $r_k$ are supposed to be known magnitudes.

Let us first treat the case where $\varpi > 0$.

In this case the method Gill, Murray and Saunders propose works as follows:

First form the vector $v_k = w_k / r_k$. After we have solved $p$ from:

$$(4.1.5) \quad Lkp = v_k,$$

the following relation holds:

$$(4.1.6) \quad A_{k+1} = L_{k+1}D_{k+1}L_{k+1}' = L_kD_kL_k' + r_k w_k w_k' = L_k(D_k + pp')L_k'$$

We now make a Cholesky decomposition of the matrix $D_k + pp'$, and we obtain:

$$(4.1.7) \quad D_k + pp' = MDM'. $$

Gill, Murray and Saunders continue by showing that the matrix $M$ is a 'special' lower triangular matrix $M(p, p, i)$. This means that:

$$(4.1.8) \quad M(i, j) = 0 \text{ for } i < j ; 1 \text{ for } i = j ; p(i)b(j) \text{ for } i > j.$$ 

The vector $p$ in formula (4.1.8) is known from equation (4.1.5), and Gill, Murray and Saunders supply an efficient forward recurrence algorithm for computing $b(j)$ $j=1,...,n$. The combination of (4.1.6) and (4.1.7) gives us an expression for the Cholesky decomposition of $A_{k+1}$:

$$(4.1.9) \quad A_{k+1} = (L_k M)(D(k M))' = L_{k+1}D_{k+1}L_{k+1}'$$

Chapter IV
Updating methods for the Cholesky decomposition.
because the product $L_kM$ is a unit lower triangular matrix. Gill, Murray and Saunders also give a forward recurrence method for computing the product of the matrices $L_k$ and $M$. When we combine this method with the algorithm for computing the variables $b(j)$, we can calculate the matrix $L_{k+1}$ directly without having to compute the matrix $M$ first.

The case $\tau < 0$

In this case we can compute a vector $p$ in a similar way as described above, so that the relation

$$(4.1.10) \quad L_{k+1}D_{k+1}L_{k+1}' = L_k(D_k-pp')L_k'$$

holds.

Proceeding in the same way we intend to compute the Cholesky decomposition of $D_k-pp'$. However, now we can not be sure that the matrix $D_k-pp'$ is positive definite. On the other hand the determinant of the matrix is:

$$(4.1.11) \quad \det(D_k-pp') = q \cdot \det(D_k),$$

where $q = 1 - p' D_k - p$, and it can be shown that a necessary and sufficient condition for the matrix $D_k-pp'$ to be positive definite, is that $q > 0$. Accordingly the second step in the algorithm of Gill, Murray and Saunders is computing the scalar $q$.

If $q$ appears to be negative, we can either stop the entire procedure and not perform a modification at all, so that $A_{k+1} = A_k$ (This is what we have done in our program), or set $q$ equal to some small constant $e$, for instance the computer precision, and proceed with the algorithm. In this case we do not perform the original modification to the matrix $A_k$, but we perform an adapted modification in order to keep the matrix positive definite.

When $q > 0$, or if we have chosen to go on with the algorithm, the rest of the algorithm is similar to the case $q > 0$, except that now backward recurrence formulae are used for computing the coefficients $b(j)$, and for multiplying the matrices $L_k$ and $M$, thus resulting in a backward recurrence algorithm.

Chapter IV

Updating methods for the Cholesky decomposition.
Section 2 Modifying the Cholesky decompositions when a constraint is dropped from the active constraint set.

Suppose the Cholesky decomposition $L_kD_kL_k^\prime$ of the matrix $N_k^T H_k^{-1} N_k$ is given, and we want to calculate the Cholesky decomposition of the matrix $N_{k+1}^T H_k^{-1} N_{k+1}$, where $N_{k+1}$ is formed from the matrix $N_k$ by deleting the $i$-th column. This change in the matrix $N_k$ amounts to deleting row $i$ and column $i$ in the matrix $N_k^T H_k^{-1} N_k$. If we analogously delete the $i$-th row in the matrix $L_k$ to obtain the matrix $L_k$, the relation:

$$L_kD_kL_k^\prime = N_{k+1}^T H_k^{-1} N_{k+1},$$

holds, but the matrix $L_k$ is not a unit lower triangular matrix, which is necessary for the Cholesky decomposition. The structure of the matrix $L_k$ is the following:

![Figure 1](image)

Here we have partitioned the matrix $L_k$ into:

Chapter IV

Updating methods for the Cholesky decomposition.
1) The \((i-1) \times (i-1)\) unit lower triangular matrix \(L_{11}\)
2) The \((q_i-1) \times (i-1)\) matrix \(L_{21}\), and
3) The \((q_i-1) \times (i+1)\) matrix \(L_{22}\).

In a similar way we can partition the matrix \(D_k\) into:
1) A \((i-1) \times (i-1)\) diagonal matrix \(D_{11}\)
2) A \((q -i+1) \times (q -i+1)\) diagonal matrix \(D_{22}\).

When we apply these partitions to the matrix \(N_k^T H_k N_k^{+}\), we obtain:

\[
(4.2.3) \quad N_k^T H_k N_k^{+} = L_k D_k L_k^T
\]

Suppose now that the real Cholesky decomposition of the matrix \(N_k^T H_k N_k^{+}\) can be written as:

\[
(4.2.4) \quad N_k^T H_k N_k^{+} = L_{k+1} D_{k+1} L_{k+1}^T,
\]

where we partition the \((q_i-1) \times (q_i-1)\) matrices \(L_{k+1}\) and \(D_{k+1}\) in a similar way as the matrices \(L_k\) and \(D_k\), we obtain:

\[
(4.2.5) \quad L_{k+1} D_{k+1} L_{k+1}^T =
\]

If we combine (4.2.3), (4.2.4) and (4.2.5) with the fact that the Cholesky decomposition is unique, we derive:

\[
(4.2.6) \quad L_{11} = L_{11}^T, \quad L_{21} = L_{21}^T, \quad D_{11} = D_{11}^T,
\]

and

\[
(4.2.7) \quad L_{22} D_{22} L_{22}^T = L_{22} D_{22} L_{22}^T.
\]

Relation (4.2.6) indicates that the first \(i-1\) columns of the matrices \(L_k\) and \(L_k\) remain unchanged. From relation (4.2.7) we can conclude that to obtain the Cholesky decomposition of \(N_k^T H_k N_k^{+}\), it suffices to compute the Cholesky decomposition of the matrix \(L_{22} D_{22} L_{22}^T\). After we have computed this decomposition, we can form the matrices \(L_{k+1}\) and \(D_{k+1}\) as:

---

Chapter IV

Updating methods for the Cholesky decomposition.
LD_{k+1} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}

\text{, and } DK_{k+1} = \begin{bmatrix} D_{11} & 0 \\ 0 & D_{22} \end{bmatrix}

\textbf{Acknowledgment}

The authors are indebted to Mrs. Anke J. Muller-Sloos for editing this report.

\textbf{Chapter IV}

\textbf{Updating methods for the holesky decomposition.}
References and Testproblems.
References.

[1] Broyden, C.G.
A class of methods for solving non-linear simultaneous equations.

[2] W. C. Davidon
Variable metric methods for minimization.
A. E. C. Research and Development report, ANL-5990

The choice of parameters in self scaling variable metric algorithms.

A rapidly convergent descent method for minimization
Computer Journal, vol 6 1963
Sequential unconstrained minimization techniques.
From Nonlinear Programming, Wiley, 1968, Chapter 4

An algorithm for quadratic programming.
Naval research logistics quarterly 3, 1965

p. 67-92

Methods for modifying matrix factorizations.
Math. of Computation, vol. 28, p. 505-535

Methods for computing and modifying the LDL-factors of a matrix.
Math. of Computation, vol. 29, p. 1051-1077

[10] D. Goldfarb
Matrix factorizations in optimization of nonlinear functions subject to linear constraints.

Extensions of Davidon's variable metric method to maximization under linear inequality and equality constraints.
A nonlinear programming technique for the optimization of continuous processing systems.
Management Science, Vol 7, July 1961

[13] Himmelblau
Applied nonlinear programming

[14] G. van der Hoek
Sequential unconstrained minimization techniques.
Lecture notes GH/402, Erasmus Univ. Rotterdam.

Experiments with a reduction method for nonlinear programming, based on a restricted lagrangian.
Report 7804, Econometric Institute, Erasmus Univ./ Rotterdam.

[16] C. L. Hooykaas and J. F. Ballintijn
VIICC, a variable metric method for linearly constrained optimization.
Shell report AMOR.0005.78, 1978, Amsterdam

VANOP, a variable metric method for nonlinear optimization.
Shell report to appear.

References.
[18] M. L. Lenard

A computational study of active set strategies in nonlinear programming with linear constraints.
NRC technical summary report #1564, Univ. of Wisconsin.

[19] D. G. Luenberger

Introduction to linear and nonlinear programming.
Addison-Wesley Publ. comp., Reading Massachusetts, (1973)


A constrained minimization method with quadratic convergence.


The least squares problem and pseudoinverses.

[22] J. E. Rosen

Two phase algorithm for nonlinear constraint problems.
Technical report 77-8, University of Minnesota.
The programs were run on the XEROX SIGMA 7 computer of Shell Research Laboratories Amsterdam.

On the following pages a set of testproblems is given on which the computer programs were tested. The set is not complete insofar that we have used many other testproblems, but because we saw no point in giving many testproblems without much variety, we have chosen some problems, that show the diverse nonlinearities we have tested. We do not give any statistics such as number of iterations and number of search directions or function evaluations. The reason for this is that those statistics are not yet available because the program continually changed until some time ago, and that the programs were written for testing the methods as to their robustness, not as to their speed. But all problems in the list have actually been solved by our programs. On the next pages you will find the testproblems.
Testproblem 1

Quadratic obj. function.
4 variables
3 linear inequality constr.
trivial constr.

Min. \( f(x) = -x_1 - 3x_2 + x_3 - x_4 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + x_2^2 + 2x_3^2 + 2x_3x_4 + x_4^2) \)

subject to
\(-x_1 - 2x_2 - x_3 - x_4 + 5 \geq 0\)
\(-3x_1 - x_2 - 2x_3 + x_4 + 4 \geq 0\)
\( x_2 + 4x_3 \geq -1.5 \geq 0\)
\( x_i \geq 0 \quad i = 1, \ldots, 4\)
\( x^0 = \left( \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4} \right) \)
\( x^* = (0.272, 2.09, 0.0, 0.545) \quad f(x^*) = -4.682 \)

Testproblem 2

Nonlinear obj. function
3 variables
2 linear inequality constr.
trivial constr.

Min. \( f(x) = \frac{2}{x_1 + \frac{1}{2}} + \frac{1}{x_2 + 0.2} + \frac{3}{x_3 + \frac{1}{2}} \)

subject to:
\(-4x_1 - 7x_2 - 3x_3 + 10 \geq 0\)
\(-3x_1 - 4x_2 - 5x_3 + 8 \geq 0\)
\( x_i \geq 0 \quad i = 1, \ldots, 3\)
\( x^0 = (-10., -10., -10.) \)
\( x^* = (0.755, 0.568, 0.691) \quad f(x^*) = 5.412 \)
Testproblem 3

Nonlinear obj. function
4 variables
8 bounds

Min. \( f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 
+ 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \)

subject to:
\(-10 \leq x_i \leq 10\)  \( i = 1, \ldots, 4 \)
\( x^0 = (-3., -1., -3., -1. ) \)
\( x^* = (1,1,1,1) \)  \( f(x^*) = 0. \) (the objective function possesses non optimal stationary points).

Testproblem 4

Nonlinear obj. function
2 variables
2 linear inequality constr.

Min. \( f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1^2) \)

subject to:
\( x_1/3 + x_2 + .1 \geq 0 \)
\(-x_1/3 + x_2 + .1 \geq 0 \)
\( x^0 = (-1.2, 1.) \)
\( x^* = (1., 1.) \)  \( f(x^*) = 0 \)
Testproblem 5

Nonlinear obj. function
3 variables
6 bounds
Min. \( f(x) = \sum_{i=1}^{99} \left[ \exp \left( \frac{-(u_i - x_2)^3}{x_1} \right) - 0.01i \right]^2 \); \( u_i = 25 + (-50 \ln 0.01.i)^{1.5} \)
subject to:
\[
0.1 < x_1 < 100. \\
0.0 < x_2 < 25.6 \\
0.0 < x_3 < 5.0 \\
x^0 = (100.0, 12.5, 3.0) \\
x^* = (50.0, 25.0, 1.5) \quad f(x^*) = 0.0
\]

Testproblem 6

Nonlinear obj. function
10 variables
20 bounds
Min. \( f(x) = \sum_{i=1}^{10} \left( \ln(x_i - 2) \right)^2 + \left( \ln(10 - x_i) \right)^2 \) \( - \prod_{i=1}^{10} x_i \) \( 0.2 \)
subject to:
\[
2.001 < x_i < 9.999 \quad i = 1, \ldots, 10 \\
x_i^0 = 9. \quad i = 1, \ldots, 10 \\
x_i^* = 9.351 \quad i = 1, \ldots, 10 \quad f(x^*) = -45.778
\]
Testproblem 7

Quadratic obj. function
3 variables
3 bounds
1 quadratic inequality

Min. \( f(x) = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2 + (x_3 - 5)^2 \)

subject to:
\[-4.5 \leq x_1 \leq 4.5\]
\[-4.5 \leq x_2 \leq 4.5\]
\[-5.0 \leq x_3 \leq 5.0\]
\[-x_1^2 - x_2^2 - x_3^2 + 48 \geq 0\]

\( x^0 = (1., 1., 1.) \)

\( x^* = (3.650, 3.650, 4.620) \quad f(x^*) = .953 \)

Testproblem 8

Quadratic obj. function
2 variables
2 nonlinear constraints
trivial constraints

Min. \( f(x) = x_1^2 + x_2^2 - 16x_1 - 10x_2 \)

subject to:
\[11 - x_1^2 + 6x_1 - 4x_2 \geq 0\]
\[x_1x_2 - 3x_2 - \exp(x_1 - 3) + 1 \geq 0\]

\( x_1 \geq 0 \)

\( x_2 \geq 0 \)

\( x^0 = (4,3) \)

\( x^* = \begin{cases} 
\text{solution of } x_1^3 - 9x_1^2 + 7x_1 + 29 + 4 \exp(x_1 - 3) = 0 \\
\text{or } x_2^* = (11 - x_1^2 + 6x_1)/4
\end{cases} \)
Testproblem 9

Quadratic obj. function
9 variables
6 bounds
12 nonlinear inequality constraints
Min. \( x_4^2 + x_5^2 + x_6^2 + x_7^2 + x_8^2 + x_9^2 \)
subject to:

\[
\begin{align*}
 x_i & \geq 0 \quad i = 4, \ldots, 9 \\
1) & \quad x_1 + x_2 \exp(-5x_3) + x_4 - 127 \geq 0 \\
2) & \quad x_1 + x_2 \exp(-3x_3) + x_5 - 151 \geq 0 \\
3) & \quad x_1 + x_2 \exp(-x_3) + x_6 - 379 \geq 0 \\
4) & \quad x_1 + x_2 \exp(x_3) + x_7 - 421 \geq 0 \\
5) & \quad x_1 + x_2 \exp(3x_3) + x_8 - 460 \geq 0 \\
6) & \quad x_1 + x_2 \exp(5x_3) + x_9 - 426 \geq 0 \\
7) & \quad -x_1 - x_2 \exp(-5x_3) + x_4 + 127 \geq 0 \\
8) & \quad -x_1 - x_2 \exp(-3x_3) + x_5 + 151 \geq 0 \\
9) & \quad -x_1 - x_2 \exp(-x_3) + x_6 + 379 \geq 0 \\
10) & \quad -x_1 - x_2 \exp(x_3) + x_7 + 421 \geq 0 \\
11) & \quad -x_1 - x_2 \exp(3x_3) + x_8 + 460 \geq 0 \\
12) & \quad -x_1 - x_2 \exp(5x_3) + x_9 + 426 \geq 0 \\
\end{align*}
\]

\( x^0 = (300.0, -100.0, -.1997, -127, -151, 379, 421, 460, 426) \)

\( x^* = : x_1 = 523.3 \\
x_2 = -156.9 \\
x_3 = -.1997 \\
x_i \text{ undetermined. } i = 4, \ldots, 9 \), not important.

Testproblem 10

Quadratic obj. function
9 variables
6 nonlinear equality constraints
Same problem as problem 9, but without bounds and with constraints 1-6 as equality constraints.
Testproblem 11

Quadratic nonconvex obj. function
9 variables
14 quadratic inequality constraints

Min. \( f(x) = -0.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_8 + x_5x_8 - x_6x_7) \)

subject to:
1. \( 1 - x_3^2 - x_4^2 \geq 0 \)
2. \( 1 - x_9 \geq 0 \)
3. \( 1 - x_5^2 - x_6^2 \geq 0 \)
4. \( 1 - x_1^2 - (x_2 - x_9)^2 \geq 0 \)
5. \( 1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0 \)
6. \( 1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0 \)
7. \( 1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0 \)
8. \( 1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0 \)
9. \( 1 - x_7^2 - (x_8 - x_9)^2 \geq 0 \)
10. \( x_1x_4 - x_2x_3 \geq 0 \)
11. \( x_3x_9 \geq 0 \)
12. \( -x_5x_9 \geq 0 \)
13. \( x_5x_8 - x_6x_7 \geq 0 \)
14. \( x_9 \geq 0 \)

\( x_i = 1 \quad i = 1, \ldots, 9 \)

\( x^* = (0.9971, -0.0758, 0.553, 0.8331, 0.9981, -0.0623, 0.5642, 0.8256, \ 0.000024) \)

\( f(x^*) = 0.8660 \)
Testproblem 12

Nonlinear obj. function
14 nonlinear inequalities
6 bounds
3 variables

Min. $f(x) = -0.063y_3y_6 + 5.04x_1 + 3.36y_4 + 0.035x_2 + 10.0x_3$

subject to:

$0 \leq y_2 \leq 5000$
$0 \leq y_3 \leq 2000$
$85 \leq y_4 \leq 93$
$90 \leq y_5 \leq 95$
$3.01 \leq y_6 \leq 12$
$.01 \leq y_7 \leq 4$
$145 \leq y_8 \leq 162$

$0 \leq x_1 \leq 2000$
$0 \leq x_2 \leq 16000$
$0 \leq x_3 \leq 120$

For the calculation of $y_i$, $i = 2, \ldots, 7$ see next page

$x^0 = (1745, 12000, 110)$

$x^* = [1728.37, 16000., 98.131$

$f(x^*) = 1162.036$
ForTRAN Description of Calculation of $Y_2 \rightarrow Y_8$

```fortran
Y(2) = 1.6*X(1)
Y(3) = 1.22*Y(2) - X(1)
Y(6) = (X(2) + Y(3))/X(1)
Y2CALC = X(1)*(112.0 + 13.167*Y(6) - 0.6667*Y(6)**2)/100.0
IF(ABS(Y2CALC - Y(2)) - 0.001)30,30,20
20 Y(2) = Y2CALC
GO TO 10
30 CONTINUE
Y(4) = 93.0
100 Y(5) = 86.35 + 1.098*Y(6) - 0.038*Y(6)**2 + 0.325*(Y(4) - 89.0)
Y(8) = -133.0 + 3.0*Y(5)
Y(7) = 35.82 - 0.222*Y(8)
Y4CALC = 98000.0*X(3)/(Y(2)*Y(7) + X(3)*1000.0)
IF(ABS(Y4CALC - Y(4)) - 0.0001)300,300,200
200 Y(4) = Y4CALC
GO TO 100
300 CONTINUE
```

Testproblems
Testproblem 13

Quadratic obj. function
3 variables
1 lin. equality constraint
1 quadratic equality constraint
trivial constraints

Min. \( f(x) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3 \)
subject to:
\[ x_1^2 + x_2^2 + x_3^2 - 25 = 0 \]
\[ 8x_1 + 14x_2 + 7x_3 - 56 = 0 \]
\[ x_i \geq 0 \quad i = 1, \ldots, 3 \]
\[ x^0 = (2.0, 2.0, 2.0) \]
\[ x^* = (3.512, .217, 3.552) \quad f(x) = 961.715 \]

Testproblem 14

Quadratic obj. function
4 variables
3 quadratic inequality constraints

Min. \( f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4 \)
subject to:
\[ x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 - 8 \leq 0 \]
\[ x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 - 10 \leq 0 \]
\[ 2x_1^2 + x_2^2 + x_3^2 + 2x_4^2 - x_2 - x_4 - 5 \leq 0 \]
\[ x^0 = (1.0, 1.0, 1.0, 1.0) \]
\[ x = (0.0, 1.0, 2.0, -1.0) \quad f(x) = -44.0 \]
Appendix A

Example for VLICO
Suppose we want to solve the following problem:

Minimize \( f(x) = 0.01(x(1) + 1)^2 + 100(x(2) - 1)^2 + \exp(2 - x(3) - x(2) + x(1)) \\
+ (x(2)^2 - x(4))^2 + \sqrt{(x(5) + 1) - 2.01} \)

subject to:
- \(-x(1) + x(3) + x(4) = 2\)
- \(x(2) + x(4) \geq 1\)
- \(-x(1) + x(2) + x(5) = 1\)
- \(-x(1) + x(2) + x(3) \leq 2\)
- \(x(2) - x(3) - x(4) + x(5) = -1\)
- \(x(3) \leq 2\)
- \(x(2) \geq .99\)
- \(x(i) \geq 0; \ i = 1, \ldots, 5\)

starting from the point: \((-1, -1, -1, -1, -1)\).

Then the problem input for VLICO will be as on the following pages. The solution VLICO gives is given after the problem inputs.

Example for VLICO

```
FUNCTION F(X, PAR, NPAR)
DIMENSION X(1)
F = .01*(X(1) + 1.)**2 + 100.*(X(2) - 1.)**2 + \
1   EXP(2. - X(3) - X(2) + X(1)) + (X(2)**2 - X(4))**2 \
2   + (X(5) + 1.)**.5 - 2.01
RETURN
END
```
Example for computer program VLICO

COL
   VAR1
   VAR2
   VAR3
   VAR4
   VAR5

ROW
   0 EQ1
   0 EQ2
   0 EQ3
   - LESS
   + MORE

MATR
   VAR1 EQ1 -1.
   VAR2 EQ2 1.
   VAR3 EQ1 1.
   VAR4 MORE 1.
   VAR3 LESS 1.
   VAR3 EQ1 1.
   VAR4 EQ1 1.
   VAR3 EQ2 -1.
   VAR4 EQ2 -1.
   VAR5 EQ2 1.
   VAR1 EQ3 -1.
   VAR2 EQ3 1.
   VAR5 EQ3 1.
   VAR1 LESS -1.
   VAR2 LESS 1.
   VAR2 MORE 1.

RHS
   EQ1 2.
   EQ2 -1.
   EQ3 1.
   LESS 2.
   MORE 1.

BOUN
   UPPR U VAR3 2.
   LOWR L VAR2 .99

INIT
   VAR1 -1.
   VAR2 -1.
   VAR3 -1.
   VAR4 -1.
   VAR5 -1.

TRIV

EOF
   .0005 .00005

The trivial constraints X(I) GE 0 are added to the constraint set.
Appendix A

Number of real parameters for function: 0
Number of integer parameters for function: 0
Number of variables = 5
Number of equality constraints = 3
Number of inequality constraints = 9

The equality constraints

Constraint EQ1

<table>
<thead>
<tr>
<th>VARIABLE VAR1 COEFFICIENT</th>
<th>-.100000E 01</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE VAR3 COEFFICIENT</td>
<td>.100000E 01</td>
</tr>
<tr>
<td>VARIABLE VAR4 COEFFICIENT</td>
<td>.100000E 01</td>
</tr>
<tr>
<td>RIGHT HAND SIDE ELEMENT</td>
<td>.200000E 01</td>
</tr>
</tbody>
</table>

Constraint EQ2

| VARIABLE VAR2 COEFFICIENT  | .100000E 01  |
| VARIABLE VAR3 COEFFICIENT  | -.100000E 01 |
| VARIABLE VAR4 COEFFICIENT  | -.100000E 01 |
| VARIABLE VAR5 COEFFICIENT  | .100000E 01  |
| RIGHT HAND SIDE ELEMENT    | -.100000E 01 |

Constraint EQ3

| VARIABLE VAR1 COEFFICIENT  | -.100000E 01 |
| VARIABLE VAR2 COEFFICIENT  | .100000E 01  |
| VARIABLE VAR5 COEFFICIENT  | .100000E 01  |
| RIGHT HAND SIDE ELEMENT    | .100000E 01  |

The inequality constraints

Constraint less

LESS THAN OR EQUAL CONSTRAINT

| VARIABLE VAR1 COEFFICIENT  | -.100000E 01 |
| VARIABLE VAR2 COEFFICIENT  | .100000E 01  |
| VARIABLE VAR3 COEFFICIENT  | .100000E 01  |
| RIGHT HAND SIDE ELEMENT    | .200000E 01  |

Constraint more

GREATER THAN OR EQUAL CONSTRAINT

| VARIABLE VAR2 COEFFICIENT  | .100000E 01  |
| VARIABLE VAR4 COEFFICIENT  | .100000E 01  |
| RIGHT HAND SIDE ELEMENT    | .100000E 01  |

Example for VLICO
Appendix A

Constraint UPPR

LESS THAN OR EQUAL CONSTRAINT
VARIABLE VAR3 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .200000E 01

Constraint LOWR

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR2 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .990000E 00

Constraint VAR1

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR1 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .000000E 00

Constraint VAR2

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR2 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .000000E 00

Constraint VAR3

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR3 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .000000E 00

Constraint VAR4

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR4 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .000000E 00

Constraint VAR5

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE VAR5 COEFFICIENT    .100000E 01
RIGHT HAND SIDE ELEMENT      .000000E 00

The initial values

VARIABLE VAR1 VALUE    -.100000E 01
VARIABLE VAR2 VALUE    -.100000E 01
VARIABLE VAR3 VALUE    -.100000E 01
VARIABLE VAR4 VALUE    -.100000E 01
VARIABLE VAR5 VALUE    -.100000E 01

Stepsize used to compute gradient approximation:  .500000E-03
Precision parameter used for convergence criterion:    .500000E-04

Example for VLICO
Example for VLICO

Appendix A

ITERATION 1  NUMBER OF VIOLATED CONSTRAINTS: 7
CONSTRAINT VAR4 NO LONGER INFEASIBLE
CONSTRAINT VAR2 NO LONGER INFEASIBLE
CONSTRAINT MORE NO LONGER INFEASIBLE
ITERATION 2
CONSTRAINT LOWR NO LONGER INFEASIBLE
CONSTRAINT VAR3 NO LONGER INFEASIBLE
ITERATION 3  NUMBER OF VIOLATED CONSTRAINTS: 2
CONSTRAINT LOWR NOW ACTIVE
ITERATION 4  NUMBER OF VIOLATED CONSTRAINTS: 2
CONSTRAINT VAR5 NO LONGER INFEASIBLE
CONSTRAINT VAR1 NO LONGER INFEASIBLE

ITERATION 1  FUNCTION VALUE: .87197161E 00
CONSTRAINT VAR1 NOW ACTIVE
ITERATION 2  FUNCTION VALUE: .35501671E 00
CONSTRAINT LESS NOW ACTIVE
ITERATION 3  FUNCTION VALUE: .15088916E -01
CONSTRAINT LOWR NO LONGER ACTIVE
ITERATION 4  FUNCTION VALUE: .15088916E -01
CONSTRAINT VAR5 NOW ACTIVE
ITERATION 5  FUNCTION VALUE: .16801059E -05
OPTIMUM FOUND AFTER 5 ITERATIONS

OPTIMUM FUNCTION VALUE IS \(0.168011\)E-05

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>ACTIVITY</th>
<th>COMPUTED FIRST DERIVATIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR1</td>
<td>0.316184E-06</td>
<td>0.101931E 01</td>
</tr>
<tr>
<td>VAR2</td>
<td>1.000000E 01</td>
<td>-0.999372E 00</td>
</tr>
<tr>
<td>VAR3</td>
<td>0.999996E 00</td>
<td>-0.999095E 00</td>
</tr>
<tr>
<td>VAR4</td>
<td>0.999991E 00</td>
<td>-0.476839E-03</td>
</tr>
<tr>
<td>VAR5</td>
<td>0.372529E-08</td>
<td>0.498772E 00</td>
</tr>
</tbody>
</table>

THE CONSTRAINTS

THE INDEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT EQ2</td>
<td>-0.999987E 00</td>
<td>-1.000000E 01</td>
<td>-1.29379E-04</td>
<td>-0.914752E-03</td>
</tr>
<tr>
<td>CONSTRAINT EQ1</td>
<td>0.200000E 01</td>
<td>0.200000E 01</td>
<td>0.000000E 00</td>
<td>-0.139171E-02</td>
</tr>
</tbody>
</table>

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT VAR1</td>
<td>0.316184E-06</td>
<td>0.000000E 00</td>
<td>-0.316184E-06</td>
<td>1.93042E-01</td>
</tr>
<tr>
<td>CONSTRAINT LESS LT</td>
<td>0.200000E 01</td>
<td>0.200000E 01</td>
<td>0.000000E 00</td>
<td>-0.998616E 00</td>
</tr>
<tr>
<td>CONSTRAINT VAR5 GT</td>
<td>0.372529E-08</td>
<td>0.000000E 00</td>
<td>-0.372529E-08</td>
<td>0.499685E 00</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT VAR2 GT</td>
<td>0.100000E 01</td>
<td>0.000000E 00</td>
<td>-1.000000E 01</td>
<td>0.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT MORE GT</td>
<td>0.199999E 01</td>
<td>1.000000E 01</td>
<td>-0.999991E 00</td>
<td>0.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT VAR4 GT</td>
<td>0.999991E 00</td>
<td>0.000000E 00</td>
<td>-0.999991E 00</td>
<td>0.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT VAR3 GT</td>
<td>0.999996E 00</td>
<td>0.000000E 00</td>
<td>-0.999996E 00</td>
<td>0.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT LOWR GT</td>
<td>0.100000E 01</td>
<td>0.990000E 00</td>
<td>-0.999999E-02</td>
<td>0.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT UPPR LT</td>
<td>0.999996E 00</td>
<td>0.200000E 01</td>
<td>-1.000000E 01</td>
<td>0.000000E 00</td>
</tr>
</tbody>
</table>

THE DEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT EQ3</td>
<td>0.100000E 01</td>
<td>1.000000E 01</td>
<td>0.000000E 00</td>
<td>0.000000E 00</td>
</tr>
</tbody>
</table>
Appendix B

Example for VANOP
Suppose we want to solve the following problem:
Minimize $f(x) = 0.1x(1)^2 + x(2)^2 + 10x(3)^2 + 100x(4)^2$
subject to:
\[ x(3) \leq -1 \]
\[ x(1) \geq 1 \]
\[ x(1) + x(2) + x(4) \geq 2 \]
\[ -x(2) + x(3) + x(4) \geq -2 \]
\[ x(2) + x(3)^2 = 3 \]
\[ (x(2) + x(3) + x(4))^2 + x(1)^2 = 3 \]
\[ x(2)^3 - x(3)^3 - x(4) \leq 10 \]
\[ x(1)^3 + x(4)^3 \leq 7 \]
\[ x(2)^2 + x(3)^2 \geq 2.5 \]
\[ x(2)x(3)x(4) - 2x(1)x(3) \geq 3 \]
starting from the point: (-10, -10, -10, -10)

Then the problem input for VANOP will be given on the following pages. The solution VANOP gives is given after the problem inputs.
FUNCTION F(X,P,NP)
DIMENSION X(1)
F=0.1*X(1)**2+X(2)**2+10.*X(3)**2+100.*X(4)**2
RETURN
END
FUNCTION CON(X,P,I)
DIMENSION X(1)
GOTO (10,20,30,40,50,60),I
10 CON=X(2)+X(3)**2-3.
RETURN
20 CON=(X(2)+X(3)+X(4)**2+X(1)**2-3.
RETURN
30 CON=X(2)**3-X(3)**3-X(4)-10.
RETURN
40 CON=X(1)**3+X(4)**3-7.
RETURN
50 CON=2.5-X(2)**2-X(3)**2
RETURN
60 CON=-X(2)*X(3)*X(4)+2.*X(1)*X(3)+3.
RETURN
END
Example for computer program VANOP

COL
  X(1)
  X(2)
  X(3)
  X(4)

ROW
  + L1
  - L2

MATR
  X(1)  L1  1.
  X(2)  L1  1.
  X(4)  L1  1.
  X(2)  L2 -1.
  X(3)  L2  1.
  X(4)  L2  1.

RHS
  L1  2.
  L2 -2.

BOUN
  UBO  U  X(3) -1.
  LOBO L  X(1)  1.

INIT
  X(1)  -10.
  X(2)  -10.
  X(3)  -10.
  X(4)  -10.

PREC
  .001
  .0001

NONL
  2  4

SUMT
  .05

NAME
  NL1
  NL2
  NL3
  NL4
  NL5
  NL6

EOF
Example for computer program VANOP

NUMBER OF REAL PARAMETERS FOR FUNCTION: 0
NUMBER OF INTEGER PARAMETERS FOR FUNCTION: 0
NUMBER OF VARIABLES: 4
NUMBER OF LINEAR EQUALITY CONSTRAINTS: 0
NUMBER OF LINEAR INEQUALITY CONSTRAINTS: 4
NUMBER OF NONLINEAR EQUALITY CONSTRAINTS: 2
NUMBER OF NONLINEAR INEQUALITY CONSTRAINTS: 4
NUMBER OF PARAMETERS FOR CONSTRAINT FUNCTION: 0

STEPSIZE USED TO COMPUTE GRADIENT APPROXIMATION: .100000E-02
PRECISION PARAMETER USED FOR CONVERGENCE CRITERION: .100000E-03
SUMT PARAMETER: .500000E-01

THE LINEAR INEQUALITY CONSTRAINTS

CONSTRAINT LIN1

GREATER THAN OR EQUAL CONSTRAINT
VARIABLE X(1) COEFFICIENT  .100000E 01
VARIABLE X(2) COEFFICIENT  .100000E 01
VARIABLE X(4) COEFFICIENT  .100000E 01
RIGHT HAND SIDE ELEMENT   .200000E 01

CONSTRAINT LIN2

LESS THAN OR EQUAL CONSTRAINT
VARIABLE X(2) COEFFICIENT  -.100000E 01
VARIABLE X(3) COEFFICIENT  .100000E 01
VARIABLE X(4) COEFFICIENT  .100000E 01
RIGHT HAND SIDE ELEMENT   -.200000E 01

CONSTRAINT UPBO

UPPER BOUND : X(3) .LE.  .100000E 01

CONSTRAINT LOBO

LOWER BOUND:  X(1) .GE.  .100000E 01

THE INITIAL VALUES

VARIABLE X(1) VALUE  -.100000E 02
VARIABLE X(2) VALUE  -.100000E 02

Example for VANOP
VARIABLE $X(3)$ VALUE $-1.00000E\ 02$
VARIABLE $X(4)$ VALUE $-1.00000E\ 02$

NAMES OF THE NONLINEAR CONSTRAINTS:

NL1
NL2
NL3
NL4
NL5
NL6

Example for VANOP
AFTER THE SUMT STEP THE PROBLEM STATISTICS ARE:
LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 36 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTIL NOW: 43

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS:  .110719E 02  PENALTY VALUE IS:  .200272E-04

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE ACTIVITY</th>
<th>COMPUTED GRADIENT</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE X(1)</td>
<td>.120707E 01</td>
<td>.241328E 00</td>
</tr>
<tr>
<td>VARIABLE X(2)</td>
<td>.989987E 00</td>
<td>.197997E 01</td>
</tr>
<tr>
<td>VARIABLE X(3)</td>
<td>-.996852E 00</td>
<td>-.199372E 02</td>
</tr>
<tr>
<td>VARIABLE X(4)</td>
<td>-.944820E -02</td>
<td>-.188950E 01</td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS

(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT LIN2</td>
<td>LT -.199629E 01</td>
<td>-.200000E 01</td>
<td>-.371202E-02</td>
<td>-.186882E 01</td>
</tr>
<tr>
<td>CONSTRAINT UPBO</td>
<td>LT -.996852E 00</td>
<td>-.100000E 01</td>
<td>-.314760E-02</td>
<td>.178415E 02</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTRAINT LIN1</td>
<td>GT .218760E 01</td>
<td>.200000E 01</td>
<td>-.187605E 00</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT LOBO</td>
<td>GT .120707E 01</td>
<td>.100000E 01</td>
<td>-.207067E 00</td>
<td>.000000E 00</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1</td>
<td>-.101630E 01</td>
<td>NL2</td>
<td>-.154272E 01</td>
<td>NL3</td>
<td>-.802971E 01</td>
</tr>
<tr>
<td>NL4</td>
<td>-.524129E 01</td>
<td>NL5</td>
<td>.526212E 00</td>
<td>NL6</td>
<td>.584142E 00</td>
</tr>
</tbody>
</table>
LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 8 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTIL NOW: 51

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS: \(0.143160 \times 10^2\) PENALTY VALUE IS: \(-0.138490 \times 10^0\)

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>ACTIVITY</th>
<th>COMPUTED GRADIENT</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(1)</td>
<td>0.185912E +01</td>
<td>0.371914E +00</td>
<td>0.417985E +00</td>
</tr>
<tr>
<td>X(2)</td>
<td>0.199627E +01</td>
<td>0.399148E +01</td>
<td>0.400961E +01</td>
</tr>
<tr>
<td>X(3)</td>
<td>-0.999262E +00</td>
<td>-0.199837E +02</td>
<td>-0.198912E +02</td>
</tr>
<tr>
<td>X(4)</td>
<td>0.547979E -03</td>
<td>0.109613E +00</td>
<td>0.224322E +00</td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS

(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)

THE INDEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>** CONSTR NL2</td>
<td>EQ</td>
<td>0.445539E +01</td>
<td>0.445714E +01</td>
<td>0.175095E -02</td>
</tr>
<tr>
<td>** CONSTR NL1</td>
<td>EQ</td>
<td>0.398836E +01</td>
<td>0.399363E +01</td>
<td>0.527000E -02</td>
</tr>
</tbody>
</table>

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>LT</td>
<td>-0.999262E +00</td>
<td>-0.100000E +01</td>
<td>-0.737548E -03</td>
<td>-0.125494E +02</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>** CONSTR NL4</td>
<td>LT</td>
<td>0.812564E +01</td>
<td>0.105170E +00</td>
<td>0.239135E +01</td>
</tr>
<tr>
<td>** CONSTR NL5</td>
<td>LT</td>
<td>-0.594478E +01</td>
<td>-0.447377E +00</td>
<td>0.147101E +01</td>
</tr>
<tr>
<td>** CONSTR NL6</td>
<td>LT</td>
<td>-0.614618E +01</td>
<td>-0.542500E +00</td>
<td>0.721188E +00</td>
</tr>
<tr>
<td>CONSTR LIN1</td>
<td>GT</td>
<td>0.385594E +01</td>
<td>0.200000E +00</td>
<td>-0.185594E +01</td>
</tr>
<tr>
<td>CONSTR LIN2</td>
<td>LT</td>
<td>-0.299499E +01</td>
<td>-0.200000E +00</td>
<td>0.994986E +01</td>
</tr>
<tr>
<td>** CONSTR NL3</td>
<td>LT</td>
<td>0.884766E +00</td>
<td>0.139215E +02</td>
<td>0.507390E +01</td>
</tr>
<tr>
<td>CONSTR LOBO</td>
<td>GT</td>
<td>0.185912E +01</td>
<td>0.100000E +01</td>
<td>-0.859124E +00</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1 =</td>
<td>-0.520325E -02</td>
<td>NL2 =</td>
<td>0.145146E +01</td>
<td>NL3 =</td>
<td>-0.104742E +01</td>
</tr>
<tr>
<td>NL4 =</td>
<td>-0.574230E +00</td>
<td>NL5 =</td>
<td>-0.248362E +01</td>
<td>NL6 =</td>
<td>-0.714412E +00</td>
</tr>
</tbody>
</table>
LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 4 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTILL NOW: 56

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS: .142625E 02 PENALTY VALUE IS: -.161295E-01

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>ACTIVITY</th>
<th>COMPUTED GRADIENT</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(1)</td>
<td>.147872E 01</td>
<td>.295676E 00</td>
<td>.895101E 00</td>
</tr>
<tr>
<td>X(2)</td>
<td>.199994E 01</td>
<td>.399915E 01</td>
<td>.405902E 01</td>
</tr>
<tr>
<td>X(3)</td>
<td>-.999993E 00</td>
<td>-.199998E 02</td>
<td>-.189375E 02</td>
</tr>
<tr>
<td>X(4)</td>
<td>-.210189E -01</td>
<td>-.420366E 01</td>
<td>-.420279E 01</td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS

(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)

THE INDEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>CONSTRAINT NL2 EQ</td>
<td>.744975E 01</td>
<td>.744979E 01</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>**</td>
<td>CONSTRAINT NL1 EQ</td>
<td>.399763E 01</td>
<td>.399768E 01</td>
<td>.000000E 00</td>
</tr>
</tbody>
</table>

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>CONSTRAINT NL6 LT</td>
<td>-.671326E 01</td>
<td>-.671334E 01</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT UPBO LT</td>
<td>-.999993E 00</td>
<td>-.100000E 01</td>
<td>.000000E 00</td>
<td>-.259913E 00</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>CONSTRAINT NL5 LT</td>
<td>-.998117E 01</td>
<td>-.748141E 01</td>
<td>.249976E 01</td>
</tr>
<tr>
<td>CONSTRAINT LIN1 GT</td>
<td>.345765E 01</td>
<td>.200000E 01</td>
<td>-.145765E 01</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>CONSTRAINT LIN2 LT</td>
<td>-.302096E 01</td>
<td>-.200000E 01</td>
<td>.102096E 01</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>**</td>
<td>CONSTRAINT NL4 LT</td>
<td>.153328E 02</td>
<td>.198513E 02</td>
<td>.451848E 01</td>
</tr>
<tr>
<td>CONSTRAINT LOBO GT</td>
<td>.147872E 01</td>
<td>.100000E 01</td>
<td>-.478724E 00</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>**</td>
<td>CONSTRAINT NL3 LT</td>
<td>.269201E 02</td>
<td>.278998E 02</td>
<td>.979660E 01</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1</td>
<td>-.696182E-04</td>
<td>NL2</td>
<td>.144930E 00</td>
<td>NL3</td>
<td>-.979677E 00</td>
</tr>
<tr>
<td>NL4</td>
<td>-.376660E 01</td>
<td>NL5</td>
<td>-.249976E 01</td>
<td>NL6</td>
<td>.537645E-03</td>
</tr>
</tbody>
</table>
LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 30 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTILL NOW: 87

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS: .143151E 02  PENALTY VALUE IS: .180435E-02

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE X(1)</th>
<th>ACTIVITY</th>
<th>01</th>
<th>COMPUTED GRADIENT</th>
<th>00</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE X(2)</td>
<td>.145861E 01</td>
<td>291695E 00</td>
<td>.315945E 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLE X(3)</td>
<td>.197768E 01</td>
<td>395506E 01</td>
<td>.399333E 01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VARIABLE X(4)</td>
<td>-.100515E 01</td>
<td>-.201065E 02</td>
<td>-.200731E 02</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS

(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)

THE INDEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL2</td>
<td>EQ</td>
<td>.615883E 01</td>
<td>.614393E 01</td>
<td>-.148954E-01</td>
</tr>
<tr>
<td>**CONSTRAINT NL1</td>
<td>EQ</td>
<td>.398744E 01</td>
<td>.399942E 01</td>
<td>.119762E-01</td>
</tr>
</tbody>
</table>

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL6</td>
<td>LT</td>
<td>-.603298E 01</td>
<td>-.604150E 01</td>
<td>-.851727E-02</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL4</td>
<td>LT</td>
<td>.956768E 01</td>
<td>.134662E 02</td>
<td>.389855E 01</td>
</tr>
<tr>
<td>**CONSTRAINT NL5</td>
<td>LT</td>
<td>-.991945E 01</td>
<td>-.749838E 01</td>
<td>.242107E 01</td>
</tr>
<tr>
<td>CONSTRAINT LIN1</td>
<td>GT</td>
<td>.340666E 01</td>
<td>.200000E 01</td>
<td>-.140665E 01</td>
</tr>
<tr>
<td>CONSTRAINT LIN2</td>
<td>LT</td>
<td>-.301247E 01</td>
<td>-.200000E 01</td>
<td>.101247E 01</td>
</tr>
<tr>
<td>CONSTRAINT UPBO</td>
<td>LT</td>
<td>-.100515E 01</td>
<td>-.100000E 01</td>
<td>.515175E-02</td>
</tr>
<tr>
<td>CONSTRAINT LOBO</td>
<td>GT</td>
<td>.145861E 01</td>
<td>.100000E 01</td>
<td>-.458609E 00</td>
</tr>
<tr>
<td>**CONSTRAINT NL3</td>
<td>LT</td>
<td>.267714E 02</td>
<td>.279940E 02</td>
<td>.122255E 01</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1 =</td>
<td>-.119896E-01</td>
<td>NL2 =</td>
<td>.165850E-01</td>
<td>NL3 =</td>
<td>-.121968E 01</td>
</tr>
<tr>
<td>NL4 =</td>
<td>-.389678E 01</td>
<td>NL5 =</td>
<td>-.242155E 01</td>
<td>NL6 =</td>
<td>.884059E-02</td>
</tr>
</tbody>
</table>
LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 5 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTIL NOW:     92

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS:     .144225E  02     PENALTY VALUE IS:     .276566E-03

THE VARIABLES

<table>
<thead>
<tr>
<th>VARIABLE ACTIVITY</th>
<th>COMPUTED GRADIENT</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARIABLE X(1)</td>
<td>.145521E  01</td>
<td>.290933E  00</td>
</tr>
<tr>
<td>VARIABLE X(2)</td>
<td>.197941E  01</td>
<td>.395869E  01</td>
</tr>
<tr>
<td>VARIABLE X(3)</td>
<td>-.101013E  01</td>
<td>-.202037E  02</td>
</tr>
<tr>
<td>VARIABLE X(4)</td>
<td>-.298259E- 01</td>
<td>-.596518E  01</td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS

(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)

THE INDEPENDENT EQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL2</td>
<td>EQ</td>
<td>.601591E  01</td>
<td>.601569E  01</td>
<td>.000000E  00</td>
</tr>
<tr>
<td>**CONSTRAINT NL1</td>
<td>EQ</td>
<td>.401045E  01</td>
<td>.401045E  01</td>
<td>.210762E- 03</td>
</tr>
</tbody>
</table>

THE ACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL6</td>
<td>LT</td>
<td>-.605054E  01</td>
<td>-.605094E  01</td>
<td>-.405312E- 03</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL4</td>
<td>LT</td>
<td>.928767E  01</td>
<td>.132061E  02</td>
<td>.391841E  01</td>
</tr>
<tr>
<td>**CONSTRAINT NL5</td>
<td>LT</td>
<td>-.985973E  01</td>
<td>-.742130E  01</td>
<td>.243843E  01</td>
</tr>
<tr>
<td>**CONSTRAINT NL3</td>
<td>LT</td>
<td>.263157E  02</td>
<td>.274996E  02</td>
<td>.118394E  01</td>
</tr>
<tr>
<td>CONSTRAINT LIN1</td>
<td>GT</td>
<td>.340480E  01</td>
<td>.200000E  01</td>
<td>-.140480E  01</td>
</tr>
<tr>
<td>CONSTRAINT LIN2</td>
<td>LT</td>
<td>-.301937E  01</td>
<td>-.200000E  01</td>
<td>.101937E  01</td>
</tr>
<tr>
<td>CONSTRAINT UPBO</td>
<td>LT</td>
<td>-.101013E  01</td>
<td>-.100000E  01</td>
<td>.101318E- 01</td>
</tr>
<tr>
<td>CONSTRAINT LOBO</td>
<td>GT</td>
<td>.145521E  01</td>
<td>.100000E  01</td>
<td>-.455214E  00</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1</td>
<td>-.222206E-03</td>
<td>NL2</td>
<td>.220597E-03</td>
<td>NL3</td>
<td>-.118399E 01</td>
</tr>
<tr>
<td>NL4</td>
<td>-.391840E 01</td>
<td>NL5</td>
<td>-.243844E 01</td>
<td>NL6</td>
<td>.449453E-03</td>
</tr>
</tbody>
</table>
TOTAL NUMBER OF MAJOR ITERATION STEPS: 5  
TOTAL NUMBER OF MINOR ITERATION STEPS: 100  

FUNCTION VALUE IS: .144244E 02  
LAST PENALTY FUNCTION VALUES WAS: .325203E - 03  

THE VARIABLES  

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>ACTIVITY</th>
<th>COMPUTED GRADIENT</th>
<th>GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X(1)</td>
<td>.145521E 01</td>
<td>.290933E 00</td>
<td>.290368E 00</td>
</tr>
<tr>
<td>X(2)</td>
<td>.197937E 01</td>
<td>.395858E 01</td>
<td>.395838E 01</td>
</tr>
<tr>
<td>X(3)</td>
<td>-.101023E 01</td>
<td>-.202076E 02</td>
<td>-.202074E 02</td>
</tr>
<tr>
<td>X(4)</td>
<td>-.298550E 01</td>
<td>-.597103E 01</td>
<td>-.597534E 01</td>
</tr>
</tbody>
</table>

THE LINEAR CONSTRAINTS  
(CONSTRAINTS MARKED WITH ** ARE LINEARIZED NONLINEAR CONSTRAINTS.)  

THE INDEPENDENT EQUALITIES  

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL2</td>
<td>EQ</td>
<td>.599899E 01</td>
<td>.599906E 01</td>
<td>.000000E 00</td>
</tr>
<tr>
<td>**CONSTRAINT NL1</td>
<td>EQ</td>
<td>.402033E 01</td>
<td>.402037E 01</td>
<td>.000000E 00</td>
</tr>
</tbody>
</table>

**CONSTRAINT NL6  

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL6</td>
<td>LT</td>
<td>-.605918E 01</td>
<td>-.605929E 01</td>
<td>.000000E 00</td>
</tr>
</tbody>
</table>

THE INACTIVE INEQUALITIES  

<table>
<thead>
<tr>
<th>TYPE</th>
<th>ACTIVITY</th>
<th>RIGHT HAND SIDE</th>
<th>SLACK ACTIVITY</th>
<th>LAGRANGE MULTIPLIER</th>
</tr>
</thead>
<tbody>
<tr>
<td>**CONSTRAINT NL4</td>
<td>LT</td>
<td>.924338E 01</td>
<td>.131618E 02</td>
<td>.391839E 01</td>
</tr>
<tr>
<td>**CONSTRAINT NL5</td>
<td>LT</td>
<td>-.987645E 01</td>
<td>-.743795E 01</td>
<td>.243850E 01</td>
</tr>
<tr>
<td>CONSTRAINT LIN1</td>
<td>GT</td>
<td>.340473E 01</td>
<td>.200000E 00</td>
<td>-.140473E 01</td>
</tr>
<tr>
<td>CONSTRAINT LIN2</td>
<td>LT</td>
<td>-.301945E 01</td>
<td>-.200000E 01</td>
<td>.101945E 01</td>
</tr>
<tr>
<td>CONSTRAINT UPB0</td>
<td>LT</td>
<td>-.101023E 01</td>
<td>-.100000E 01</td>
<td>.102262E 01</td>
</tr>
<tr>
<td>CONSTRAINT LOB0</td>
<td>GT</td>
<td>.145521E 01</td>
<td>.100000E 01</td>
<td>-.455209E 00</td>
</tr>
<tr>
<td>**CONSTRAINT NL3</td>
<td>LT</td>
<td>.263876E 02</td>
<td>.275716E 02</td>
<td>.118401E 01</td>
</tr>
</tbody>
</table>

VALUES OF THE NONLINEAR CONSTRAINTS ARE:  

<table>
<thead>
<tr>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
<th>CONSTRAINT NAME</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL1</td>
<td>-.696182E-04</td>
<td>NL2</td>
<td>-.975132E-04</td>
<td>NL3</td>
<td>-.118412E 01</td>
</tr>
<tr>
<td>NL4</td>
<td>-.391843E 01</td>
<td>NL5</td>
<td>-.243848E 01</td>
<td>NL6</td>
<td>.121638E-03</td>
</tr>
<tr>
<td>Report Number</td>
<td>Title</td>
<td>Authors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7800</td>
<td>&quot;List of Reprints, nos 200-208; Abstracts of Reports Second Half 1977&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7802/S</td>
<td>&quot;General Quadratic Forms in Normal Variates&quot;, by C. Dubbelman.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7803/S</td>
<td>&quot;On Bahadur's Representation of Sample Quantiles&quot;, by L. de Haan and E. Taconis-Haantjes.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7805/S</td>
<td>&quot;Derivatives of Regularly Varying Functions in IR^d and Domains of Attraction of Stable Distributions&quot;, by L. de Haan and S.I. Resnick.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7807/E</td>
<td>&quot;Money Illusion and Aggregation Bias&quot;, by J. van Daal.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7810/M</td>
<td>&quot;A New Proof of Cartier's Third Theorem&quot;, by M. Hazewinkel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7811/M</td>
<td>&quot;On the (Internal) Symmetry Groups of Linear Dynamical Systems&quot;, by M. Hazewinkel.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7814/E</td>
<td>&quot;An Improvement of the Main Program from Report 7304&quot;, by C. Dubbelman.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7815</td>
<td>&quot;Publications of the Econometric Institute, First Half 1978: List of Reprints, Nos. 209-219; Abstracts of Reports.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7818/M</td>
<td>&quot;Note on the Eigenvalues of the Covariance Matrix of Disturbances in the General Linear Model&quot;, by R.J. Stroeker.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7820/S</td>
<td>&quot;An Abel-Tauber Theorem on Convolutions with the Möbius Functions&quot;, by J. Geluk.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

"An Abel-Tauber Theorem on Convolutions with the Möbius Functions", by J. Geluk.

"Optimization Methods Based on Projected Variable Metric Search Directions", by J.F. Ballintijn, G. van der Hoek and C.L. Hooykaas.