Netherlands school of economics

ECONOMETRIC INSTITUTE

OPTIMIZATION METHODS BASED ON PROJECTED VARIABLE METRIC SEARCH

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

Erasmus

REPORT 7821/O

ERASMUS UNIVERSITY ROTTERDAM, P.O. BOX 1738, ROTTERDAM, THE NETHERLANDS

OPTIMIZATION METHODS BASED ON PROJECTED VARIABLE METRIX SEARCH DIRECTIONS

J.F. Ballintijn[1], G. van der Hoek[2], C.L. Hooykaas[1]

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 behaviour 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 [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.

Introduction.

Minimize $f(\underline{x})$

Subject to :

$\underline{a}_i^T \cdot \underline{x} = b_i \qquad i=1,2,.....m_1$

$\underline{a}_i^T \cdot \underline{x} \leq b_i \qquad i=m_1+1,.....,m$

Here $f(\underline{x})$ 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, $\underline{x}_k$, the search direction, $\underline{p}_k$, is computed using information on the gradient, $\nabla f(\underline{x}_k)$, and some approximation of the hessian, $\nabla^2 f(\underline{x}_k)$, 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 $f(\underline{x})$ is a quadratic function quasi Newton (=variable metric) methods converge in at most n steps, where n is the dimension of the vector $\underline{x}$.

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

[10], 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].

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:

$$(1.1)\quad \text{Minimize } q(\underline{x}) = .5 * \underline{x}^T \cdot A \cdot \underline{x} + \underline{b}^T \cdot \underline{x} + c \ ,$$

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

$$(1.2)\quad \nabla q(\underline{x}) = A \cdot \underline{x} + \underline{b} \ ,$$

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

$$(1.3)\quad \nabla q(\underline{x}_1) - \nabla q(\underline{x}_2) = A \cdot (\underline{x}_1 - \underline{x}_2) \ .$$

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

$$(1.4)\quad \nabla q(\underline{x}^*) = \underline{0} \ .$$

From (1.3) and (1.4) it now follows that :

$$(1.5) \quad \nabla q(\underline{x}) = A\,(\underline{x} - \underline{x}^*) \text{ , or}$$

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

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 $\underline{x}^*$ by adding a perturbance matrix to the matrix A, to obtain $\tilde{A}$ and consequently compute $\tilde{A}^{-1}$ and use this matrix in relation (1.6). According to relation (1.6) the minimum ,$\underline{x}^*$, of a quadratic function can be calculated if in a point ,$\underline{x}$, the gradient ,$\nabla f(\underline{x})$, and the inverse of the hessian matrix ,H (or $\nabla^2 f(\underline{x})=A$ 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 ,$f(\underline{x})$, will not be a quadratic function. For this case convergence has also been proven [19], when started with an initial estimate of $\underline{x}^*$ which is sufficiently close to the optimum, and when the function $f(\underline{x})$ 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].

The variable metric methods are based on the following ideas:

i) An approximation H_k to the hessian H and an estimate $\underline{x}_k$ of the optimum $\underline{x}^*$ are given.

ii) According to (1.6) the direction, $\underline{p}_k$, in which the optimum will be looked for is calculated by:

$$-H_k^{-1} \cdot \nabla f(\underline{x}_k).$$

Because H_k is only an approximation to the hessian matrix H, a line search along the search direction $\underline{p}_k$ 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 α_k is the value of α minimizing the objective function $f(\underline{x})$ along the line $\underline{x}_k + \alpha \underline{p}_k$, then we can calculate the next iteration point as:

$$(1.7) \quad \underline{x}_{k+1} = \underline{x}_k + \alpha_k \cdot \underline{p}_k .$$

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

$$(1.8) \quad H \cdot (\underline{x}_{k+1} - \underline{x}_k) = \nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k)$$

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

$$(1.9) \quad H_{k+1} = H_k + C_k \text{ , and}$$

$$(1.10) \quad H_{k+1} \cdot (\underline{x}_{k+1} - \underline{x}_k) = \nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k) ,$$

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^{-1} to H^{-1}, and update this estimate in accordance with the relation:

$$(1.11)\quad \underline{x}_{k+1} - \underline{x}_k = H^{-1}\cdot\{\nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k)\}\ ,$$

to obtain :

$$(1.12)\quad \underline{x}_{k+1} - \underline{x}_k = H_{k+1}^{-1}\ \{\nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k)\}.$$

From the many different possibilities to update the matrix H_k or H_k^{-1} 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 :

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

ii) Less computational work is required.

iii) 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 :

(2.1) Minimize $f(\underline{x})$

Subject to :

$$\underline{a}_i^T \cdot \underline{x} = b_i \qquad i = 1, \ldots\ldots, m_1$$

$$\underline{a}_i^T \cdot \underline{x} \leq b_i \qquad i = m_1 + 1, \ldots\ldots, m$$

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

Given a feasible initial point $\underline{x}_1$, both methods determine which constraints are active in $\underline{x}_1$, and form a matrix N_1 of full rank, whose columns are the q_1 linearly independent normals of these active constraints.

Goldfarb's idea was to start with an approximation H_1^{-1} of the inverse hessian matrix, that has the property :

(2.2) $N_1^T \cdot H_1^{-1} = \emptyset$.

Chapter II
Quasi newton methods with linear constraints.

Thus the matrix H_1^{-1} is not of full rank. It then holds for the first search direction, $\underline{p}_1$:

(2.3) $\underline{p}_1 = -H_1^{-1} \cdot \nabla f(\underline{x}_1)$, that

(2.4) $N_1^T \cdot \underline{p}_1 = -N_1^T \cdot H_1^{-1} \nabla f(\underline{x}_1) = \underline{0}$.

This implies that the search direction $\underline{p}_1$ is parallel to the hyperspace spanned by the active constraints. Now for the update of the approximate inverse hessian H_k^{-1} update formulæ are chosen in such a way that whenever no changes in the active constraint set occur the following implication holds

(2.5) $N_k^T \cdot H_k^{-1} = 0 \Rightarrow N_{k+1}^T \cdot H_{k+1}^{-1} = N_k^T \cdot H_{k+1}^{-1} = 0$.

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 charge, the relation :

(2.6) $N_k^T \cdot \underline{p}_k = \underline{0}$, where $\underline{p}_k = -H_k^{-1} \cdot \nabla f(\underline{x}_k)$,

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^{-1}, given a set of active constraints and corresponding matrix N_1 , is according to [7], the projection matrix :

(2.7) $H_1^{-1} = I - N_1 (N_1^T N_1)^{-1} N_1^T$

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

For the case where a new constraint with corresponding normal vector $\underline{n}_T$ is added to N_k this implies that H_{k+1}^{-1} has to be formed in such a way that :

(2.8) $H_{k+1}^{-1} \underline{n}_T = \underline{0}$.

Chapter II
Quasi newton methods with linear constraints.

This condition ensures that the search direction $\underline{p}_{k+1}$ 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 $\underline{n}_T$,

$$(2.9) \qquad H_{k+1}^{-1}\,\underline{n}_T \neq \underline{0} \ ,$$

should hold. This condition, which increases the rank of the matrix H_k^{-1} with one, is necessary to ensure that the search direction $\underline{p}_{k+1}$ 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_k^{-1} 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 :

$$(2.10) \qquad \underline{p}_k^* = -H_k^{-1}\,\nabla f(\underline{x}_k)$$

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

$$(2.11) \qquad P_k = I - H_k^{-1} N_k (N_k^T H_k^{-1} N_k)^{-1} N_k^T$$

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 $(N_k^T H_k^{-1} N_k)^{-1}$ after a quasi Newton correction in H_k , thus avoiding the need to recalculate $(N_k^T H_k^{-1} N_k)^{-1}$ 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 $(N_k^T \cdot H_k^{-1} \cdot N_k)^{-1}$ are available. The algorithm of Murtagh and Sargent is :

Chapter II
Quasi newton methods with linear constraints.

STEP 1 Take a feasible initial point $\underline{x}_1$ (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 $\underline{x}_1$, 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 :

$$\underline{p}_k^* = -H_k^{-1} \nabla f(\underline{x}) \tag{2.12}$$

and premultiply $\underline{p}_k^*$ with the matrix :

$$P_k = I - H_k^{-1} N_k (N_k^T H_k^{-1} N_k)^{-1} N_k^T \tag{2.13}$$

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

$$\underline{p}_k = P_k \underline{p}_k^* \tag{2.14}$$

The set of Lagrange multipliers corresponding to the active constraints, can be calculated during the computation of $\underline{p}_k$ [15]. The Lagrange multipliers are given by

$$\underline{\lambda}_k = (N_k^T H_k^{-1} N_k)^{-1} N_k^T H_k^{-1} \nabla f(\underline{x}_k) \tag{2.15}$$

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 $\|\underline{p}_k\| \leq \varepsilon$ Stop.

Ctherwise go on to step 4.

STEP 4 Execute a minimization along the search direction $\underline{p}_k$, taking into account that the maximum step size, $\bar{\alpha}_k$, which can be taken is equal to the distance from the current iteration point to the nearest constraint along the direction $\underline{p}_k$:

$$(2.16) \qquad \bar{\alpha}_k = \min_i \{(b_i - \underline{a}_i^T \underline{x}_k)/\underline{a}_i^T \underline{p}_k \mid \underline{a}_i^T \underline{p}_k > 0\},$$

where $\underline{a}_i$ is the normal of the i-th constraint. Set $\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k$ where α_k is the minimzing 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 occured.

STEP 6 If the step size α_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 $(N_k^T H_k^{-1} N_k)^{-1}$ 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 computercode 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^{-1} 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 decompcsitions 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 $v.v^T$ 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 $v.v^T$. 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 herefore [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 $N_{k+1}^T H_{k+1}^{-1} N_{k+1} = N_{k+1}^T H_k^{-1} N_{k+1}$.

Chapter III
Description of the implemented algorithm.

No method to calculate the Cholesky factors L_{k+1} and D_{k+1} of the matrix $N_{k+1}^T \cdot H_{k+1}^{-1} \cdot N_{k+1}$ 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 $\underline{x}_1$, with its gradient $\nabla f(\underline{x}_1)$. Take the unit matrix I as the first approximation H_1 to the hessian matrix of the objective function $f(\underline{x})$. Determine the set of active constraints and the corresponding matrix N consisting of the normals of the active constraints. Compute the Cholesky decomposition of $N_1^T \cdot H_1^{-1} \cdot N_1$, and set the iteration counter k=1.

Step 2 Determine the search direction :

$$\underline{p}_k = -P_k H_k^{-1} \cdot \nabla f(\underline{x}_k) ,$$

where $P_k = I - H_k^{-1} N_k (N_k^T H_k^{-1} N_k)^{-1} N_k^T$

and compute the maximum steplength $\bar{\alpha}_k$ along $\underline{p}_k$, so that for $0 < \alpha < \bar{\alpha}_k$, $\underline{x}_k + \alpha \underline{p}_k$ is a feasible point. (for an exact formulation of $\bar{\alpha}_k$ see formula (2.16)). Also calculate the approximation of the lagrange multipliers:

$$\underline{\lambda}_k = (N_k^T H_k^{-1} N_k)^{-1} N_k^T H_k^{-1} \cdot \nabla f(\underline{x}_k)$$

If $\|N_k^T \cdot p_k\| > \varepsilon$, where ε is a small user supplied constant, the search direction is not parallel to the active constraints; Go to step 6.
If $\|p_k\| < \varepsilon$, or if in iteration k-1 the set of

Chapter III
Description of the implemented algorithm.

active constraints was changed, go to step 7. Otherwise go to step 3.

Step 3 Find the steplength α_k ($0 < \alpha_k \leq \bar{\alpha}_k$) that minimizes $f(\underline{x}_k + \alpha \underline{p}_k)$.

Step 4 Set $\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k$, and modify the Cholesky decompositions of H_k and $N_k^T \cdot H_k^{-1} \cdot N_k$ to obtain H_{k+1} and $N_{k+1}^T H_{k+1}^{-1} N_{k+1}$. If $\alpha_k = \bar{\alpha}_k$, 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 $N_k^T \cdot H_k^{-1} N_k$ 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 $N_k^T H_k^{-1} N_k$ to $N_k^T N_k$.

Step 7 Select the largest lagrange multiplier $\lambda(j)$ and calculate $\beta = .5\, \lambda(j) / b(jj)$, where $b(jj)$ is the j-th diagonal element of $(N_k^T \cdot H_k^{-1} N_k)^{-1}$. β can be interpreted as the expected improvement when constraint j is dropped from the set of active constraints.
Stop the procedure if both $\|\underline{p}_k\| < \varepsilon$ and $\beta < \varepsilon$. If $\|\underline{p}_k\| \leq \beta$ 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 $N_{k+1}^T \cdot H_{k+1}^{-1} \cdot N_{k+1}$. Set $\underline{x}_{k+1} = \underline{x}_k$ 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 $\underline{x}_0$ 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 ν be the set of indices i for which the constraints :

$$\underline{a}_i^T \cdot \underline{x} \leq b_i$$

are violated. Then we choose as penalty function:

$$p(\underline{x}) = \sum_{i \in \nu} \{\underline{a}_i^T \cdot \underline{x} - b_i + .1 * (1. + |b_i|)\}^2$$

We now use the above described algorithm to solve a problem with $p(\underline{x})$ 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 ν if any constraints did become feasible.
- Because the set ν can change every iteration, we do not have the same function every iteration, and it does not make sense to update the matrices H_k and $N_k^T H_k^{-1} N_k$ 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.

Chapter III
Description of the implemented algorithm.

Part II

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

Introduction.

In recent years the field of nonlinear programming has received very much attention. One reason for this may be that thrcugh 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) Minimize $f(\underline{x})$

Subject to :

$h_j(\underline{x}) = 0 \qquad j=1,.....,mA$

$g_i(\underline{x}) \leq 0 \qquad i=1,.....,mB$

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

The MAP code transforms this problem to a sequence of linear programming problems, generated by linearization of toth the nonlinear objective function $f(\underline{x})$ and the nonlinear constraints $h_j(\underline{x})$, $j=1,.....,mA$ and $g_i(\underline{x})$, $i=1,.....,mB$, 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:

$$m(\underline{x}) = f(\underline{x}) + \sum_{j=1}^{m_A} \lambda_j [\, h_j(\underline{x}) - l_j(\underline{x},\underline{y}) \,] + \sum_{i=1}^{m_B} \mu_i [\, g_i(\underline{x}) - k_i(\underline{x},\underline{y}) \,]$$

Where λ_j and μ_i are the Lagrange multipliers of the nonlinear constraints h_j and g_i, and $l_j(\underline{x},\underline{y})$ and $k_i(\underline{x},\underline{y})$ are the linearizations of these constraints with respect to the point $\underline{y}$.

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 robuster convergence. Where MAP generates a sequence of LP problems, the proposed method creates a series of problems of the form:

(1.2) minimize $m(\underline{x})$

Subject to :

$\underline{a}_j^T \cdot \underline{x} = b_j \quad j=1,.....m1$

$\underline{a}_i^T \cdot \underline{x} \leq b_i \quad i=m1+1,.....m2$

Where $m(\underline{x})$ is a nonlinear function.

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.

Introduction.

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 S1 and S2 as follows:

$$(1.1)\quad S1 = \left\{ \underline{x} \;\middle|\; \begin{array}{l} \underline{a}_i^T \underline{x} - b_i = 0 \;;\; i=1,\ldots m1 \\ \underline{a}_i^T \underline{x} - b_i \leq 0 \;;\; i=m1+1,\ldots m2 \end{array} \right\}$$

S1 is the feasible region defined by the linear constraints only, and:

$$(1.2)\quad S2 = \left\{ \underline{x} \;\middle|\; \begin{array}{l} h_j(\underline{x}) = 0 \;\; j=1,\ldots m3 \\ g_i(\underline{x}) \leq 0 \;\; i=1,\ldots m4 \end{array} \right\}$$

S2 is the feasible region defined by the nonlinear constraints only

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

(1.3) Minimize $f(\underline{x})$

subject to :

$\underline{x} \in S1 \cap S2$

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

Chapter I
Theoretical backgrounds.

(1.4) Minimize $m(\underline{x}) = f(\underline{x}) + s(\underline{x},\underline{x}_{k-1})$

subject to :

$\underline{x} \in S1 \cap T2(\underline{x}_{k-1})$

where $T2(\underline{x}_{k-1})$ is the region created by linearization of the nonlinear constraints:

$$(1.5)\quad T2(\underline{x}_{k-1}) = \left\{ \underline{x} \;\middle|\; \begin{array}{l} l_j(\underline{x},\underline{x}_{k-1}) = 0 \;;\; j=1,\ldots m3 \\ k_i(\underline{x},\underline{x}_{k-1}) = 0 \;;\; i=1,\ldots m4 \end{array} \right\}$$

where the linearization of $h_j(\underline{x})$, $l_j(\underline{x},\underline{x}_{k-1})$, is defined as:

$$(1.6)\quad l_j(\underline{x},\underline{x}_{k-1}) = h(\underline{x}) + (\underline{x} - \underline{x}_{k-1})^T \nabla h_j(\underline{x}_{k-1})$$

and $k_i(\underline{x},\underline{x}_{k-1})$ is defined in the same way as linearization of $g_i(x)$. Rosen [22] has proved that if a solution $\underline{x}_k$ to (1.4) is a fixed point of the mapping of $E^n \rightarrow E^n$ defined by the algorithm then $\underline{x}_k$ is also a Kuhn-Tucker point of problem (1.3). The convergence of the series $\{\underline{x}_k\}$ to a solution of (1.3) of course depends heavily on the function $s(\underline{x},\underline{x}_{k-1})$. Further we want the function $s(\underline{x},\underline{x}_{k-1})$ to possess the following properties:

$$(1.7)\quad s(\underline{x}_{k-1},\underline{x}_{k-1}) = 0,$$

$$(1.8)\quad \nabla s(\underline{x}_{k-1},\underline{x}_{k-1}) = \underline{0}.$$

Because when $\underline{x}_k = \underline{x}_{k-1}$, i.e. the solution to (1.3) is found, we want the following relations to hold:

$$(1.9)\quad m(\underline{x}_k) = f(\underline{x}_k), \text{ and}$$

$$(1.10)\quad \nabla m(\underline{x}_k) = \nabla f(\underline{x}_k).$$

The modified Lagrange function is an objective function $m(\underline{x})$ that possesses properties (1.9) and (1.10), and of which fast convergence is proven for the sequence $\{\underline{x}_k\}$.
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:

$$(1.11)\quad s(\underline{x},\underline{x}_{k-1}) = \sum_{i=1}^{m_3} \lambda_i(\underline{x}_{k-1})\cdot\{h_i(\underline{x}) - l_i(\underline{x}_{k-1},\underline{x})\} + \sum_{j=1}^{m_4} \mu_j(\underline{x}_{k-1})\cdot\{g_j(\underline{x}) - k_j(\underline{x},\underline{x}_{k-1})\}$$

where $\lambda_i(\underline{x}_{k-1})$ and $\mu_j(\underline{x}_{k-1})$ are approximations to the Lagrange multipliers of the constraints $h_i(\underline{x})$ and $g_j(\underline{x})$ in the point $\underline{x}_{k-1}$. Note that the function $s(\underline{x},\underline{x}_{k-1})$ contains information concerning the nonlinearity of the constraints. Convergence proofs for this objective function exist, when the starting point $\underline{x}_1$ is 'sufficiently close' to a local minimum of (1.3). Starting from an arbitrary point $\underline{x}_1$, however, no convergence can be guaranteed.

To overcome this difficulty, we introduce a first phase to obtain a good starting point $\underline{x}_1$, with corresponding estimates of the Lagrange multipliers $\lambda_i(\underline{x}_1)$ and $\mu_j(\underline{x}_1)$. The method we have used for this first phase, is solving the problem:

$$(1.12)\quad \text{Minimize} \quad f(\underline{x}) + p(\underline{x})$$

subject to :

$$\underline{x} \in S1,$$

where $p(\underline{x})$ is defined as:

$$(1.13)\quad p(\underline{x}) = .5\cdot\pi_1\cdot\left[\sum_{j=1}^{m_3}\{h_j(\underline{x})\}^2 + \sum_{i=1}^{m_4}\{g_i^+(\underline{x})\}^2\right],$$

where π_1 is a penalty parameter, and $g_i^+(\underline{x})$ is defined as:

$$(1.14)\quad g_i^+(\underline{x}) = \max.\ \{0,\ g_i(\underline{x})\}$$

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 $\{\underline{x}_k\}$ that converges to a local minimum, by increasing the value of the penalty parameter π_k 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_1 the solution to problem (1.12) also gives good approximations of the Lagrange multipliers $\lambda_i(\underline{x}_1)$ and $\mu_j(\underline{x}_1)$ in $\underline{x}_1$ given as:

Chapter I
Theoretical backgrounds.

$$(1.15) \qquad \lambda_i(\underline{x}_1) = \pi_1 \cdot h_i(\underline{x}_1)$$

$$(1.16) \qquad \mu_j(\underline{x}_1) = \pi_1 \cdot g_j^+(\underline{x}_1).$$

It can be shown [22], that if π_1 is chosen greater than some constant, the sequence $\{\underline{x}_k\}$ 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 $\underline{x}_o$; the linear constraints ; the nonlinear constraints $h_i(\underline{x})$ and $g_j(\underline{x})$; Parameters for VLICO and VANOP , such as precision parameters and penalty parameters.

Step 1 Solve with VLICO the problem :

(2.1) Minimize $f(\underline{x}) + p(\underline{x})$,

subject to $\underline{x} \in S1$,

where S1 is defined in (1.1) and $p(\underline{x})$ in (1.13), starting in point $\underline{x}_o$ to obtain $\underline{x}_1$ and $\underline{\lambda}_1$. Set $\underline{x}_k = \underline{x}_1$, $\underline{\lambda}_k = \underline{\lambda}_1$ and k=1. If no point $\underline{x} \in S1$ can be found stop because it is an infeasble problem.

Step 2 Given $\underline{x}_k$ and $\underline{\lambda}_k$ generate the feasible region $T2(\underline{x}_k)$ as defined in (1.5).

Step 3 Solve with VLICO the problem :

(2.2) Minimize $f(\underline{x}) + s(\underline{x},\underline{x}_k)$

subject to $\underline{x} \in T2(\underline{x}_k) \cap S1$

where $s(\underline{x},\underline{x}_k)$ is given in (1.11), to obtain the vectors $\underline{x}_{k+1}$ and $\underline{\lambda}_{k+1}$.

Step 4 If $\|\underline{x}_{k+1} - \underline{x}_k\| \leq \varepsilon$, where ε is some predetermined constant, stop because the method has converged.

Step 5 Set $\underline{x}_{k+1} = \underline{x}_k$; $\underline{\lambda}_{k+1} = \underline{\lambda}_k$ 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 computerprogram 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.

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*n ($m \geq n$) matrix l(i,j), for which the following relation holds:

l(i,j) = Ø for j=i+1 to n, and i=1 to n.

In a picture :

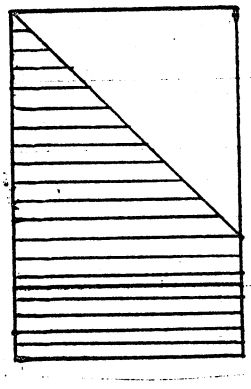

figure 1

2. A lower triangular matrix L is a n*n matrix l(i,j) for which the following relation holds:

l(i,j)=Ø for j=i+1 to n, and i=1 to n.

In a picture :

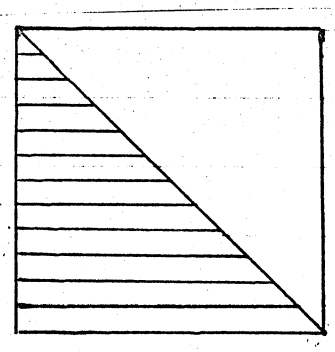

figure 2

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,c) is a triangular matrix m(i,j) for which the following relations hold:

m(i,j)=0 for j=i+1 to n, and i=1 to n

m(i,j)=c(i) for j=i, and i=1 to n.

m(i,j)=p(i)b(j) for j=1 to i-1, and i=1 to n.

6. An elementary matrix Ek is a n*n triangular matrix e(i,j), for which the following relations hold:

e(i,j)=1 for j=i, and i=1 to n

e(i,j)=0 for j=i+1 to n, and i=1 to n

e(i,j)=0 for i=j+1 to n, and j=1 to k-1

e(i,j)=0 for i=j+1 to n, and j=k+1 to n

In a picture :

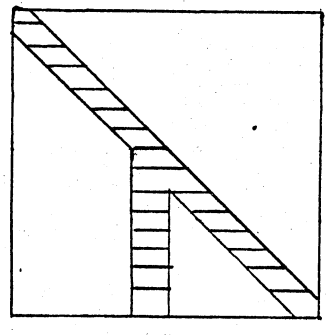

figure 3

7. A diagonal matrix D is a n*n square matrix d(i,j), for which the following relation holds:

$d(i,j)=\emptyset$ for $j \neq i$, and $i,j=1$ to n

In a picture :

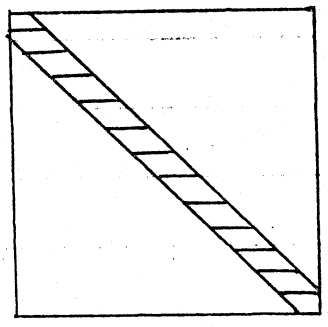

figure 4

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

9. A permutation matrix P is a n*n matrix p(i,j), for which the following relations hold :

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

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

$p(i,j)=0$ 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 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*n ($m \geq n$) matrix A consists of a m*n lower trapezoidal matrix L and an n*n unit upper triangular matrix U, such that:

$$A = LU \qquad (2.1.1)$$

The elements of a Cholesky decomposition of a positive definite symmetric matrix B, are a n*n unit lower triangular matrix L,

and a diagonal matrix D, such that:

$$B = LDL^T \qquad (2.1.2)$$

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.

Chapter II
Decomposition methods for matrices.

To compute the inverse of a nonsingular n*n matrix A, for example, is very simple if the LU-factorization (2.1.1) of this matrix is known. The matrix A^{-1} is then equal to $A^{-1} = U^{-1}\cdot L^{-1}$ and the matrices U^{-1} and L^{-1} 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) $A\cdot\underline{x} = \underline{b}$

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

(2.1.4) $LDL^{T}\cdot\underline{x} = \underline{b}$

We can solve $\underline{x}$ from (2.1.4) by solving the following sequence:

(2.1.5) solve $\underline{y}$ from $L\underline{y} = \underline{b}$

(2.1.6) compute $\underline{w} = D^{-1}\cdot\underline{y}$

(2.1.7) solve $\underline{x}$ from $L^{T}\underline{x} = \underline{w}$

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 $\underline{w}$ 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 n*m matrix (m>n), we make a LU-factorization of A and solve only the first n variables of $\underline{x}$ 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.

Chapter II
Decomposition methods for matrices.

Section 2 A method for LU-decomposition.

This method [21] for finding a m*n lower trapezoidal matrix L and a n*n unit upper triangular matrix U ,such that: A=LU , generates a sequence L0U0, L1U1, L2U2,....., LnUn, where L=Ln and U=Un, and the relation A=LiUi always holds; for a practical example see section 4 of this chapter.

Start with L0=A and U0=I, and form the elementary matrix E1 such that for the matrix L1=L0.E1 it holds that the elements L1(1,j) for j>1 are zero. Now compute the matrix $E1^{-1}$ (This is also an elementary matrix E1^.) and premultiply U0 with this matrix to obtain U1=E1^.U0.

Now L1.U1=L0.E1.E1^.U0=A.I.I=A. After this we form in the same way matrices:

(2.2.1) Lk=L0.E1.E2.....Ek , and

(2.2.2) Uk=Ek^.Ek-1^.....E1^.U0

Where every time the matrices E1 to Ek, and E1^ to Ek^ are chosen in such a way that the relation:

Lk(i,j)=0 for j>i, and i≤k ,

holds for the matrix Lk, and Ei.Ei^=I for i=1 to k. Now for the matrix Uk it then holds that :

Uk(i,i)=1 , and

Uk(i,j)≠0 only for i<j and i≤k,

and all other elements are equal to zero. Proceeding in this way we obtain L=Ln and U=Un.

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) P=Pn.Pn-1.....P1 ,

where each matrix Pk is a matrix that interchanges two rows in a matrix. Now each matrix Pk is chosen in such a way that in the matrix

Chapter II
Decomposition methods for matrices.

$$L^*k-1=Pk.Pk-1....P1.L\emptyset.E1.....Ek-1 ,$$

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:

$$(2.2.4) \quad .5*mn^2 - n^3/6 + \mathcal{O}(n^2)$$

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*n symmetric positive definite matrix An can be written as:

$$(3.3.2) \quad An = \left[\begin{array}{c|c} An-1 & \underline{b} \\ \hline \underline{b}^T & ann \end{array}\right]$$

where we know the Cholesky decomposition of the (n-1)*(n-1) matrix An-1 :

$$(3.3.3) \quad An-1 = Ln-1.Dn-1.Ln-1'$$

Then we can find the Cholesky-factors Ln and Dn of the matrix An in the following way:

If we put :

$$Ln = \left[\begin{array}{c|c} Ln-1 & \emptyset \\ \hline \underline{c}^T & 1 \end{array}\right] , \text{ and } Dn = \left[\begin{array}{c|c} Dn-1 & \emptyset \\ \hline \emptyset & x \end{array}\right]$$

Then we derive the following relation:

$$(3.3.4)\quad \left[\begin{array}{c|c} An-1 & \underline{b} \\ \hline \underline{b}^T & ann \end{array}\right] = An = Ln.Dn.Ln^T = \left[\begin{array}{c|c} Ln-1.Dn-1.Ln-1^T & Ln-1.Dn-1.\underline{c} \\ \hline \underline{c}'Dn-1.Ln-1^T & \underline{c}'Dn-1.\underline{c}+x \end{array}\right]$$

Now we can compute $\underline{c}$ from the relation

$$(3.3.5)\quad Ln-1Dn-1.\underline{c} = \underline{b}\ ,$$

by a backsubstitution because Ln-1 is a triangular matrix. We can compute x from:

$$(3.3.6)\quad \underline{c}'Dn-1.\underline{c}+x = a_{nn}.$$

The relation between the positive definiteness of An and the sign of x can easily be derived :

If An is positive definite, then so is An-1, and:

$$0<\det(An) = \det(Ln).\det(Dn).\det(Ln')=\det(Dn-1).x\ ,$$

so x is positive because both det(Dn) and det(Dn-1) are positive. Now the Cholesky factors can easily be computed using the above derived recurrence relations, because L1=1, and D1=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.A.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:

$$(2.4.1)\quad A = \begin{bmatrix} 1 & 2 & 0 \\ 5 & 3 & 2 \\ 4 & 1 & 3 \\ 2 & 1 & 5 \\ 0 & 2 & 2 \end{bmatrix}$$

Then we will have to start with the following matrices L0 and U0 :

$$L0 = A \text{ , and } U0 = I.$$

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 L0 by the elementary matrix E1, and premultiply U0 by $E1^{-1}$, where E1 and $E1^{-1}$ are:

$$(2.4.2)\quad E1 = \begin{bmatrix} 1 & -3/5 & -2/5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{ and } E1^{-1} = \begin{bmatrix} 1 & 3/5 & 2/5 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And we obtain : $U1 = E1^{-1}$, and :

$$(2.4.3)\quad L1 = \begin{bmatrix} 5 & 0 & 0 \\ 1 & 7/5 & -2/5 \\ 4 & -7/5 & 7/5 \\ 2 & -1/5 & 21/5 \\ 0 & 2 & 2 \end{bmatrix}$$

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 L1 by the matrix E2, and premultiply U1 by $E2^{-1}$, where E2 and $E2^{-1}$ are given by:

$$(2.4.4)\quad E2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} ; \quad E2^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

After this step we obtain :

$$(2.4.5)\quad L2 = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 2 & 0 \\ 4 & -7/5 & 14/5 \\ 2 & -1/5 & 22/5 \\ 1 & 7/5 & -9/5 \end{bmatrix}, \text{ and } U2 = \begin{bmatrix} 1 & 3/5 & 2/5 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

For the case of numerical stability, we now interchange row 3 and row 4 in L2 to find the matrix L3=L to find the matrix U we do not have to change the matrix U2 anymore. The product of the matrices L and U now gives the matrix A with a modified row sequence :

$$(2.4.6)\quad \begin{bmatrix} 5 & 3 & 2 \\ 0 & 2 & 2 \\ 2 & 1 & 5 \\ 4 & 1 & 3 \\ 1 & 2 & 0 \end{bmatrix}$$

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 the stability [illegible], we will first have to order the matrix on its diagonal elements, and we obtain:

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

Now we can start making the Cholesky decomposition. We first take I1=1 and D1=B(1,1)=3. We now know that:

Chapter II
Decomposition methods for matrices.

$$(2.5.3)\quad L2 = \begin{bmatrix} 1 & 0 \\ c1 & 1 \end{bmatrix}, \text{ and } D2 = \begin{bmatrix} 3 & 0 \\ 0 & x2 \end{bmatrix}$$

And we can solve c1 from the equation :

$$(2.5.4)\quad 3c1 = 3/2 \Rightarrow c1 = 1/2$$

and x2 from :

$$(2.5.5)\quad 3/4 + x2 = 11/4 \Rightarrow x2 = 2$$

Our third step consists of computing L and D as:

$$(2.5.6)\quad L = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ c1 & c2 & 1 \end{bmatrix}, \text{ and } D = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & x3 \end{bmatrix}$$

If we take $\underline{c} = (c1, c2)^T$, we can compute $\underline{c}$ from $L2D2.\underline{c} = (1, 13/10)^T$, this corresponds with:

$$(2.5.7)\quad 3c1 = 1 \text{ , and}$$

$$(2.5.8)\quad 3c1/2 + 2c2 = 13/10.$$

and solution of these two equations gives c1=1/3, and c2=2/5. We can now calculate x3 from:

$$(2.5.8)\quad \underline{c}'D2\underline{c} + x3 = 124/75 \Rightarrow x3 = 1$$

We now have the matrices L and D such that the product $LDL' = B$.

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 Ak, and we want to derive the simplest correction matrix Ck such that the matrix:

$$(3.1.1) \quad Ak+1 = Ak + Ck \ ,$$

satisfies the equation :

$$(3.1.2) \quad Ak+1 \cdot (\underline{x}_{k+1} - \underline{x}_k) = \nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k).$$

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

$$(3.1.3) \quad \underline{z}_k = \underline{x}_{k+1} - \underline{x}_k \ , \text{ and}$$

$$(3.1.4) \quad \underline{y}_k = \nabla f(\underline{x}_{k+1}) - \nabla f(\underline{x}_k) \ , \text{ and}$$

$$(3.1.5) \quad \underline{v}_k = \underline{y}_k - Ak\underline{z}_k$$

inserting (3.1.1) in (3.1.2) yields :

$$(3.1.6) \quad Ck\underline{z}_k = \underline{y}_k .$$

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

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

$$(3.1.7) \quad Ck = r_k \underline{w}_k \underline{w}_k^T ,$$

where $\underline{w}_k$ 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 $\underline{w}_k$ are uniquely determined.

Inserting (3.1.7) in (3.1.6) yields :

$$(3.1.8) \quad r_k \underline{w}_k \underline{w}_k' \underline{z}_k = \underline{y}_k \text{ ,thus}$$

$$(3.1.9) \quad \underline{w}_k = q_k \underline{y}_k ,$$

where $q_k = 1/(r_k \underline{w}_k' \underline{z}_k)$ is some unknown scalar. Using (3.1.9) on its turn in relation (3.1.8) yields:

$$(3.1.10) \quad r_k q_k^2 \underline{y}_k \underline{y}_k' \underline{z}_k = \underline{y}_k$$

from which expression we can derive :

$$(3.1.11) \quad r_k q_k^2 = 1/(\underline{y}_k' \underline{z}_k)$$

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

$$(3.1.12) \quad Ck = r_k q_k^2 \underline{y}_k \underline{y}_k' = \underline{y}_k \underline{y}_k' / (\underline{y}_k' \underline{z}_k)$$

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 Bk and let the k+1-th approximation be given as:

(3.1.13) $B_{k+1} = B_k + D_k$.

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

(3.1.14) $\underline{z}_k = B_{k+1}\,\underline{y}_k$,

r if we use the notation :

(3.1.15) $\underline{s}_k = \underline{z}_k - B_k\underline{y}_k$,

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

(3.1.16) $D_k\underline{y}_k = \underline{s}_k$.

or the rank one update formula we can write Dk as:

(3.1.17) $D_k = a_k\underline{t}_k\underline{t}_k'$,

here a_k is a scalar and $\underline{t}_k$ a n-dimensional vector. ubstituting (3.1.17) in (3.1.16) gives :

(3.1.18) $a_k\underline{t}_k\underline{t}_k'\underline{y}_k = \underline{s}_k$, or

(3.1.19) $\underline{t}_k = e_k\underline{s}_k$,

here $e_k = 1/(a_k\underline{t}_k'\underline{y}_k)$ is some unknown scalar. Substituting .1.19) in (3.1.18) gives:

(3.1.20) $a_k e_k^2\underline{s}_k\underline{s}_k'\underline{y}_k = \underline{s}_k$,

nd we can conclude that :

(3.1.21) $a_k e_k^2 = 1/(\underline{s}_k'\underline{y}_k)$.

nserting (3.1.19) into (3.1.17), and using (3.1.21) now ves us the desired expression for Dk :

(3.1.22) $D_k = a_k e_k^2\underline{s}_k\underline{s}_k' = \underline{s}_k\underline{s}_k'/(\underline{s}_k'\underline{y}_k)$

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 Ak and Bk for the Hessian matrix and the inverse of the Hessian matrix respectively, such that BkAk = I, this relation will also hold for the matrices Ak+1 and Bk+1 if the rank one correction formulae, which have been derived in the first section of this chapter, are used.

Suppose we have:

$$(3.2.1) \quad B_kA_k = I$$

$$(3.2.2) \quad B_{k+1} = B_k + \underline{s}_k\underline{s}_k'/(\underline{s}_k'\underline{y}_k) \text{ , and}$$

$$(3.3.3) \quad A_{k+1} = A_k + \underline{v}_k\underline{v}_k'/(\underline{v}_k'\underline{z}_k) \text{ ,}$$

then we can write $B_{k+1}A_{k+1}$ as :

$$(3.2.4) \quad B_{k+1}A_{k+1} = B_kA_k + \underline{s}_k\underline{s}_k'A_k/(\underline{s}_k'\underline{y}_k) + B_k\underline{v}_k\underline{v}_k'/(\underline{v}_k'\underline{z}_k) + \underline{s}_k\underline{s}_k'\underline{v}_k\underline{v}_k'/(\underline{s}_k'\underline{y}_k\underline{v}_k'\underline{z}_k)$$

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

$$(3.2.5) \quad \underline{s}_k'A_k = (\underline{z}_k' - \underline{y}_k'B_k)A_k = \underline{z}_k'A_k - \underline{y}_k = -\underline{v}_k' \text{ ,and}$$

$$(3.2.6) \quad B_k\underline{v} = B_k(\underline{y}_k - A\cdot\underline{z}_k) = B_k\underline{y}_k - \underline{z}_k = -\underline{s}_k$$

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

$$(3.2.7) \quad B_{k+1}A_{k+1} = I + \underline{s}_k\underline{v}_k'\{\underline{s}_k'\underline{v}_k/(\underline{s}_k'\underline{y}_k\underline{v}_k'\underline{z}_k) - 1/(\underline{s}_k'\underline{y}_k) - 1/(\underline{v}_k'\underline{z}_k)\}$$

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

$$(3.2.8) \quad \{(\underline{s}_k'\underline{v}_k - \underline{s}_k'\underline{y}_k - \underline{v}_k'\underline{z}_k)/(\underline{s}_k'\underline{y}_k\underline{v}_k'\underline{z}_k)\}$$

Using equations (3.1.5) and (3.1.15) gives the relations:

(3.2.9) $\underline{s}_k'\underline{y}_k = (\underline{z}_k - Bk\underline{y}_k)'(\underline{y}_k - Ak\underline{z}_k) =$

$= 2\underline{z}_k'\underline{y}_k - \underline{y}_k^T Bk\underline{y}_k - \underline{z}_k'Ak\underline{z}_k$,

(3.2.10) $\underline{s}_k'\underline{y}_k = (\underline{z}_k - Bk\underline{y}_k)'\underline{y}_k = \underline{z}_k'\underline{y}_k - \underline{y}_k^T Bk\underline{y}_k$,

(3.2.11) $\underline{y}_k'\underline{z}_k = (\underline{y}_k - Ak\underline{z}_k)'\underline{z}_k = \underline{z}_k'\underline{y}_k - \underline{z}_k'Ak\underline{z}_k$.

Inserting these relations in (3.2.8) reduces the term between brackets to zero, and from this fact follows that $Bk+1Ak+1 = I$, 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 Ak 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 :

$$(4.1.1) \quad Ck = r_k \underline{w}_k \underline{w}_k'$$

where r_k is a scalar and $\underline{w}_k$ is a n-dimensional vector. As in our program we store the matrix Ak in the form of its Cholesky decomposition:

$$(4.1.2) \quad Ak = LkDkLk' \text{ , where Lk is a unit lower triangular matrix, and Dk is a diagonal matrix, we need a set of modification formulae for the Cholesky factors Lk and Dk which correspond with the correction of the matrix Ak:}$$

$$(4.1.3) \quad Ak+1 = Ak + r_k \underline{w}_k \underline{w}_k' .$$

In [8] Gill, Murray and Saunders show simple recurrence relations which yield the Cholesky factors Lk+1 and Dk+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 Lk and Dk, the vector $\underline{w}_k$ and the scalar r_k are supposed to be known magnitudes.

Let us first treat the case where $r_k > 0$.

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

First form the vector $\underline{v}_k = \underline{w}_k / r_k$. After we have solved $\underline{p}$ from:

$$(4.1.5) \quad L_k \underline{p} = \underline{v}_k ,$$

the following relation holds:

$$(4.1.6) \quad A_{k+1} = L_{k+1} D_{k+1} L_{k+1}' = L_k D_k L_k' + r_k \underline{w}_k \underline{w}_k' =$$

$$= L_k D_k L_k' + \underline{v}_k \underline{v}_k' = L_k (D_k + \underline{p}\underline{p}') L_k'$$

We now make a Cholesky decomposition of the matrix $D_k + \underline{p}\underline{p}'$, and we obtain:

$$(4.1.7) \quad D_k + \underline{p}\underline{p}' = M D M' .$$

Gill, Murray and Saunders continue by showing that the matrix M is a 'special' lower triangular matrix $M(\underline{p}, \underline{b}, \underline{i})$. 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 $\underline{p}$ 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 Ak+1:

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

Chapter IV
Updating methods for the Cholesky decomposition.

because the product LkM is a unit lower triangular matrix. Gill, Murray and Saunders also give a forward recurrence method for computing the product of the matrices Lk and M. When we combine this method with the algorithm for computing the variables b(j), we can calculate the matrix Lk+1 directly without having to compute the matrix M first.

The case $r_k \leq 0$

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

(4.1.10) $Lk+1 \cdot Dk+1 \cdot Lk+1' = Lk(Dk-pp')Lk'$ holds.

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

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

where $q=1-p' \cdot Dk \cdot p$, and it can be shown that a necessary and sufficient condition for the matrix Dk-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 Ak+1=Ak (This is what we have done in our program), or set q equal to some small constant e, for instance the computerprecision, and proceed with the algorithm. In this case we do not perform the original modification to the matrix Ak, 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 $r_k>0$, except that now backward recurrence formulae are used for computing the coefficients b(j), and for multiplying the matrices Lk 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 $L_k D_k L_k'$ of the matrix $N_k^T H_k^{-1} N$ 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 $\tilde{L}_k$, the relation:

$$(4.2.1) \qquad \tilde{L}_k D_k \tilde{L}_k' = N_{k+1}^T H_k^{-1} N_{k+1} ,$$

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

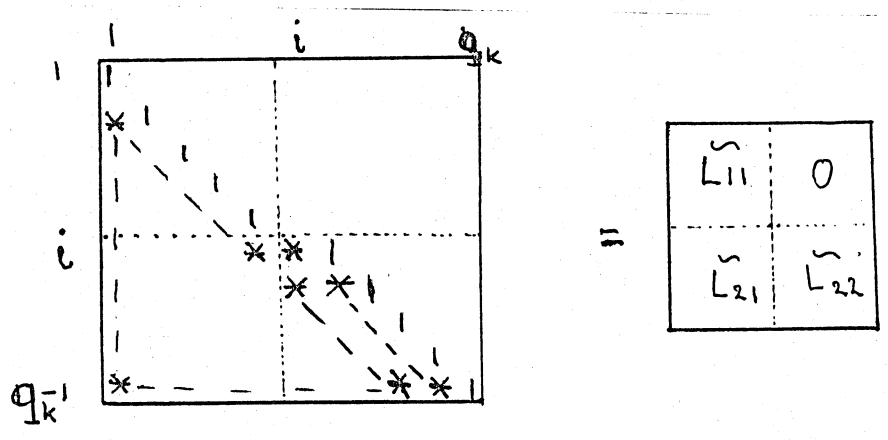

figure 1

Here we have partitioned the matrix L_k into:

Chapter IV
Updating methods for the Cholesky decomposition.

i) The (i-1)*(i-1) unit lower triangular matrix $\tilde{L}11$

ii) The $(q_k-i)*(i-1)$ matrix $\tilde{L}21$, and

iii) The $(q_k-i)*(q_k-i+1)$ matrix $\tilde{L}22$.

In a similar way we can partition the matrix Dk into:

i) A (i-1)*(i-1) diagonal matrix $\tilde{D}11$

ii) A $(q-i+1)*(q-i+1)$ diagonal matrix $\tilde{D}22$.

When we apply these partitions to the matrix $N_{k+1}^T H_k^{-1} N_{k+1}$ we ottain:

$$(4.2.3)\quad N_{k+1}^T H_k^{-1} N_{k+1} = \tilde{L}k D k \tilde{L}k = \left[\begin{array}{c|c} \tilde{L}_{11}\tilde{D}_{11}\tilde{L}_{11}^T & \tilde{L}_{21}\tilde{D}_{11}\tilde{L}_{11}^T \\ \hline \tilde{L}_{11}\tilde{D}_{11}\tilde{L}_{21}^T & \tilde{L}_{21}\tilde{D}_{11}\tilde{L}_{21}^T + \tilde{L}_{22}\tilde{D}_{22}\tilde{L}_{22}^T \end{array}\right]$$

Suppose now that the real Cholesky decomposition of the matrix $N_{k+1}^T H_k^{-1} N_{k+1}$ can be written as:

$$(4.2.4)\quad N_{k+1}^T H_k^{-1} N_{k+1} = Lk+1\,Dk+1\,Lk+1' ,$$

where we partition the $(q_k-1)*(q_k-1)$ matrices Lk+1 and Dk+1 in a similar way as the matrices Lk and Dk, we obtain:

$$(4.2.5)\quad Lk+1\,Dk+1\,Lk+1 = \left[\begin{array}{c|c} L_{11}D_{11}L_{11}^T & L_{21}D_{11}L_{11}^T \\ \hline L_{11}D_{11}L_{21}^T & L_{21}D_{11}L_{21}^T + L_{22}D_{22}L_{22}^T \end{array}\right]$$

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

$$(4.2.6)\quad L11=\tilde{L}11 \;;\; L21=\tilde{L}21 \text{ and } D11=\tilde{D}11 ,$$

and

$$(4.2.7)\quad L22\,D22\,L22' = \tilde{L}22\,\tilde{D}22\,\tilde{L}22'$$

Relation (4.2.6) indicates that the first i-1 columns of the matrices Lk and Dk remain unchanged. From relation (4.2.7) we can conclude that to ottain the Cholesky decomposition of $N_{k+1}^T H_k^{-1} N_{k+1}$ it suffices to compute the Cholesky decomposition of the matrix $L22\,D22\,L22'$. After we have computed this decomposition, we can form the matrices Lk+1 and Dk+1 as:

Chapter IV
Updating methods for the Cholesky decomposition.

$$L_{k+1} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \quad \text{, and } D_{k+1} = \begin{bmatrix} D_{11} & 0 \\ 0 & D_{22} \end{bmatrix}$$

Acknowledgment

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

Chapter IV
Updating methods for the holesky decomposition.

References and Testproblems.

References.

[1] Broyden, C.G.

A classof methods for solving non-linear simultaneous equations.

Maths. Comput. 19, pp 577-593, 1965

[2] W. C. Davidon

Variable metric methods for minimization.

A. E. C. Research and Development report, ANL-5990

[3] M. W. Dijkshoorn and G. van der Hoek

The choice of parameters in self scaling variable metric algorithms.

Report Econometric Inst., Erasmus Univ. Rotterdam, to appear 1978.

[4] R. Fletcher and M. J. D. Powell

A rapidly convergent descent method for minimization

Computer Journal, vol 6 1963

References.

[5] A. V. Fiacco and G. P. McCormick

Sequential unconstrained minimization techniques.

From Nonlinear Programming, Wiley, 1968, Chapter 4

[6] M. Frank and P. Wolfe

An algorithm for quadratic programming.

Naval research logisics quarterly 3, 1965

[7] P. E. Gill and W. Murray

Numerical methods for constrained optimization, Acad. Press, New York, 1974.

p. 67-92

[8] P. E. Gill, W. Murray, M. Saunders, G. H. Golub

Methods for modifying matrix factorizations.

Math. of Computation, vol. 28, p. 505-535

[9] P. E. Gill, W. Murray and M. Saunders

Methods for computing and modifying the LDV-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.

Math. programming 10, p. 1-31

[11] D. Goldfarb

Extensions of Davidons variable metric method to maximization under linear inequality and equality constraints.

SIAM journal of appl. math., vol. 17, p. 739-764

References.

[12] R. E. Griffith and R. A. Stewart

A nonlinear programming technique for the optimization of continuous processing systems.

Mangement Science, Vol 7, july 1961

[13] Himmelblau

Applied nonlinear programming

McGraw-Hill, 1972.

[14] G. van der Hoek

Sequential unconstrained minimization techniques.

Lecture notes GH/402, Erasmus Univ. Rotterdam.

[15] G. van der Hoek

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

VLICO, a variable metric method for linearly constrained optimization.

Shell report AMOR.0005.78, 1978, Amsterdam

[17] C. L. Hooykaas and J. F. Ballintijn

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.

MRC technical summary report #1564, Univ. of Wisconsin.

[19] D. G. Luenberger

Introduction to linear and nonlinear programming.

Addison-Wesley Publ. comp., Reading Massachusets,(1973)

[20] B.A. Murtagh and R.W.H. Sargent

A constrained minimization method with quadratic convergence.

In: R. Fletcher, Optimization, Acad. Press, New York 1969. p. 215-246

[21] G. Peters and J. H. Wilkinson

The least squares problem and pseudoinverses.

The Computer Journal, 13, pp. 309-316.

[22] J. B. Rosen

Two phase algorithm for nonlinear constraint problems.

Technical report 77-8, University of Minnesota.

TESTPROBLEMS

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.

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

subject to

$$-x_1 - 2x_2 - x_3 - x_4 + 5 \geq 0$$

$$-3x_1 - x_2 - 2x_3 + x_4 + 4 \geq 0$$

$$x_2 + 4x_3 - 1.5 \geq 0$$

$$x_i \geq 0 \quad i = 1, \ldots, 4$$

$$x^0 = (\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2}, \tfrac{1}{2})$$

$$x^* = (.272, 2.09, .0, .545) \quad f(x^*) = -4.682$$

Testproblem 2

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

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

subject to:

$$-4x_1 - 7x_2 - 3x_3 + 10 \geq 0$$

$$-3x_1 - 4x_2 - 5x_3 + 8 \geq 0$$

$$x_i \geq 0 \quad i = 1, \ldots, 3$$

$$x^0 = (-10., -10., -10.)$$

$$x^* = (.755, .568, .691) \quad f(x^*) = 5.412$$

Testproblem 3

Nonlinear obj. function
4 variables
8 bounds

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

subject to:

$-10 \leq x_i \leq 10 \quad i = 1, \ldots, 4$

$x^0 = (-3., -1., -3., -1.)$

$x^* = (1,1,1,1) \quad f(x^*) = 0.$ (the objective function possesses non optimal stationary points).

Testproblem 4

Nonlinear obj. function
2 variables
2 linear inequality constr.

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

subject to:

$x_1/3 + x_2 + .1 \geq 0$

$-x_1/3 + x_2 + .1 \geq 0$

$x^0 = (-1.2, 1.)$

$x^* = (1., 1.) \quad f(x^*) = 0$

Testproblem 5

Nonlinear obj. function
3 variables
6 bounds

$$\text{Min. } f(x) = \sum_{i=1}^{99} \left[\exp\left(\frac{-(u_i - x_2)^{x_3}}{x_1} \right) - 0.01i \right]^2 \quad ; \quad u_i = 25 + (-50 \ln 0.01.i)^{\frac{1}{1.5}}$$

subject to:

$0.1 \leq x_1 \leq 100.$

$0.0 \leq x_2 \leq 25.6$

$0.0 \leq x_3 \leq 5.0$

$x^0 = (100.0, 12.5, 3.0)$

$x^* = (50.0, 25.0, 1.5) \qquad f(x^*) = 0.0$

Testproblem 6

Nonlinear obj. function
10 variables
20 bounds

$$\text{Min. } f(x) = \sum_{i=1}^{10} ((\ln(x_i - 2))^2 + (\ln(10 - x_i))^2) - \left(\prod_{i=1}^{10} x_i \right)^{0.2}$$

subject to:

$2.001 \leq x_i \leq 9.999 \quad i = 1, \ldots, 10$

$x_i^0 = 9. \quad i = 1, \ldots, 10$

$x_i^* = 9.351 \quad i = 1, \ldots, 10 \quad f(x^*) = -45.778$

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

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:

$x_i \geq 0 \quad i = 4, \ldots, 9$

1) $x_1 + x_2 \exp(-5x_3) + x_4 - 127 \geq 0$

2) $x_1 + x_2 \exp(-3x_3) + x_5 - 151 \geq 0$

3) $x_1 + x_2 \exp(-x_3) + x_6 - 379 \geq 0$

4) $x_1 + x_2 \exp(x_3) + x_7 - 421 \geq 0$

5) $x_1 + x_2 \exp(3x_3) + x_8 - 460 \geq 0$

6) $x_1 + x_2 \exp(5x_3) + x_9 - 426 \geq 0$

7) $-x_1 - x_2 \exp(-5x_3) + x_4 + 127 \geq 0$

8) $-x_1 - x_2 \exp(-3x_3) + x_5 + 151 \geq 0$

9) $-x_1 - x_2 \exp(-x_3) + x_6 + 379 \geq 0$

10) $-x_1 - x_2 \exp(x_3) + x_7 + 421 \geq 0$

11) $-x_1 - x_2 \exp(3x_3) + x_8 + 460 \geq 0$

12) $-x_1 - x_2 \exp(5x_3) + x_9 + 426 \geq 0$

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

$x^* = : x_1 = 523.3$
$x_2 = -156.9$
$x_3 = -.1997$
x_i 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. $f(x) = -.5(x_1x_4 - x_2x_3 + x_3x_9 - x_5x_9 + x_5x_8 - x_6x_7)$

subject to:

$1 - x_3^2 - x_4^2 \geq 0$

$1 - x_9^2 \geq 0$

$1 - x_5^2 - x_6^2 \geq 0$

$1 - x_1^2 - (x_2 - x_9)^2 \geq 0$

$1 - (x_1 - x_5)^2 - (x_2 - x_6)^2 \geq 0$

$1 - (x_1 - x_7)^2 - (x_2 - x_8)^2 \geq 0$

$1 - (x_3 - x_5)^2 - (x_4 - x_6)^2 \geq 0$

$1 - (x_3 - x_7)^2 - (x_4 - x_8)^2 \geq 0$

$1 - x_7^2 - (x_8 - x_9)^2 \geq 0$

$x_1x_4 - x_2x_3 \geq 0$

$x_3x_9 \geq 0$

$-x_5x_9 \geq 0$

$x_5x_8 - x_6x_7 \geq 0$

$x_9 \geq 0$

$x_i^0 = 1 \quad i = 1, \ldots, 9$

$x^* = (.9971, -.0758, .553, .8331, .9981, -.0623, .5642, .8256, .0000024)$

$f(x^*) = .8660$

Testproblem 12

Nonlinear obj. function
14 nonlinear inequalities
6 bounds
3 variables

Min. $f(x) = -.063y_3y_6 + 5.04x_1 + 3.36y_4 + .035x_2 + 10.0x_3$

subject to:

$0 \leq y_2 \leq 5000$

$0 \leq y_3 \leq 2000$

$85 \leq y_4 \leq 93$

$90 \leq y_5 \leq 95$

$3. \leq y_6 \leq 12$

$.01 \leq y_7 \leq 4$

$145 \leq y_8 \leq 162$

$0 \leq x_1 \leq 2000$

$0 \leq x_2 \leq 16000$

$0 \leq x_3 \leq 120$

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

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

$x^* = [1728.37, 16000., 98.13]$

$f(x^*) = 1162.036$

Fortran Description of Calculation of $Y_2 \rightarrow Y_8$

```
      Y(2) = 1.6*X(1)
10    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

$$\text{Min. } f(x) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3$$

subject to:

$$x_1^2 + x_2^2 + x_3^2 - 25 = 0$$

$$8x_1 + 14x_2 + 7x_3 - 56 = 0$$

$$x_i \geq 0 \quad i = 1, \ldots, 3$$

$$x^0 = (2., 2., 2.)$$

$$x^* = (3.512, .217, 3.552) \qquad f(x) = 961.715$$

Testproblem 14

Quadratic obj. function

4 variables

3 quadratic inequality constraints

$$\text{Min. } f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$

subject to:

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 - 8 \leq 0$$

$$x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 - 10 \leq 0$$

$$2x_1^2 + x_2^2 + x_3^2 + 2x_4^2 - x_2 - x_4 - 5 \leq 0$$

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

$$x = (0., 1., 2., -1.) \quad f(x) = -44.$$

Appendix A

Example for VLICO

Suppose we want to solve the following problem:

Minimize $f(x) = .01(x(1) + 1)^2 + 100(x(2) - 1)^2 + \exp(2 - x(3) - x(2) + x(1))$
$+ (x(2)^2 - x(4))^2 + \sqrt{(x(5) + 1)} - 2.01$

subject to:

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

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.

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

VARIABLE VAR1 COEFFICIENT	-.100000E 01
VARIABLE VAR3 COEFFICIENT	.100000E 01
VARIABLE VAR4 COEFFICIENT	.100000E 01
RIGHT HAND SIDE ELEMENT	.200000E 01

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

```
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
```

```
ITERATION      1        NUMBER OF VIOLATED CONSTRAINTS: 7
CONSTRAINT   VAR4    NO LONGER INFEASIBLE
CONSTRAINT   VAR2    NO LONGER INFEASIBLE
CONSTRAINT   MORE    NO LONGER INFEASIBLE
ITERATION      2        NUMBER OF VIOLATED CONSTRAINTS: 4
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
```

Example for VLICO

OPTIMUM FOUND AFTER 5 ITERATIONS

OPTIMUM FUNCTION VALUE IS .168011E-05

THE VARIABLES

	VARIABLE ACTIVITY	COMPUTED FIRST DERIVATIVE
VARIABLE VAR1	.316184E-06	.101931E 01
VARIABLE VAR2	.100000E 01	-.999372E 00
VARIABLE VAR3	.999996E 00	-.999095E 00
VARIABLE VAR4	.999991E 00	-.476839E-03
VARIABLE VAR5	.372529E-08	.498772E 00

THE CONSTRAINTS

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT EQ2	EQ	-.999987E 00	-.100000E 01	-.129379E-04	-.914752E-03
CONSTRAINT EQ1	EQ	.200000E 01	.200000E 01	.000000E 00	-.139171E-02

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT VAR1	GT	.316184E-06	.000000E 00	-.316184E-06	.193042E-01
CONSTRAINT LESS	LT	.200000E 01	.200000E 01	.000000E 00	-.998616E 00
CONSTRAINT VAR5	GT	.372529E-08	.000000E 00	-.372529E-08	.499685E 00

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT VAR2	GT	.100000E 01	.000000E 00	-.100000E 01	.000000E 00
CONSTRAINT MORE	GT	.199999E 01	.100000E 01	-.999991E 00	.000000E 00
CONSTRAINT VAR4	GT	.999991E 00	.000000E 00	-.999991E 00	.000000E 00
CONSTRAINT VAR3	GT	.999996E 00	.000000E 00	-.999996E 00	.000000E 00
CONSTRAINT LOWR	GT	.100000E 01	.990000E 00	-.999999E-02	.000000E 00
CONSTRAINT UPPR	LT	.999996E 00	.200000E 01	.100000E 01	.000000E 00

THE DEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT EQ3	EQ	.100000E 01	.1000000E 01	.000000E 00	.000000E 00

Appendix B

Example for VANOP

Suppose we want to solve the following problem:

Minimize $f(x) = .1x(1)^2 + x(2)^2 + 10x(3)^2 + 100x(4)^2$

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.

Example for VANOP

```
      FUNCTION F(X,P,NP)
      DIMENSION X(1)
      F=.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 VANOP

Example for computer program VANOP

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

ROW
              +  LIN1
              -  LIN2

MATR
          X(1)  LIN1   1.
          X(2)  LIN1   1.
          X(4)  LIN1   1.
          X(2)  LIN2   -1.
          X(3)  LIN2   1.
          X(4)  LIN2   1.

RHS
                LIN1   2.
                LIN2   -2.

BOUN
       UPBO 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
            0

SUMT
.05
NAME
                NL1
                NL2
                NL3
                NL4
                NL5
                NL6

EOF
```

Example for VANOP

Example for computerprogram 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   -.100000E  02
VARIABLE X(4) VALUE   -.100000E  02
```

NAMES OF THE NONLINEAR CONSTRAINTS:

NL1
NL2
NL3
NL4
NL5
NL6

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

	VARIABLE ACTIVITY	COMPUTED GRADIENT	GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS
VARIABLE X(1)	.120707E 01	.241328E 00	.244418E 00
VARIABLE X(2)	.989987E 00	.197997E 01	.196948E 01
VARIABLE X(3)	-.996852E 00	-.199372E 02	-.199374E 02
VARIABLE X(4)	-.944820E-02	-.188950E 01	-.190015E 01

THE LINEAR CONSTRAINTS

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

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT LIN2	LT	-.199629E 01	-.200000E 01	-.371202E-02	-.186882E 01
CONSTRAINT UPB0	LT	-.996852E 00	-.100000E 01	-.314760E-02	-.178415E 02

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT LIN1	GT	.218760E 01	.200000E 01	-.187605E 00	.000000E 00
CONSTRAINT LOB0	GT	.120707E 01	.100000E 01	-.207067E 00	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.101630E 01	NL2 =	-.154272E 01	NL3 =	-.802971E 01
NL4 =	-.524129E 01	NL5 =	.526212E 00	NL6 =	.584142E 00

********** PROBLEM STATISTICS

LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 8 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTILL NOW: 51

FUNCTION VALUE AFTER THIS MAJOR ITERATION IS: .143160E 02 PENALTY VALUE IS : -.138490E 00

THE VARIABLES

	VARIABLE ACTIVITY	COMPUTED GRADIENT	GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS
VARIABLE X(1)	.185912E 01	.371914E 00	.417985E 00
VARIABLE X(2)	.199627E 01	.399148E 01	.400961E 01
VARIABLE X(3)	-.999262E 00	-.199837E 02	-.198912E 02
VARIABLE X(4)	.547979E-03	.109613E 00	.224322E 00

THE LINEAR CONSTRAINTS

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

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
** CONSTRAINT NL2	EQ	.445539E 01	.445714E 01	.175095E-02	.112418E 00
** CONSTRAINTS NL1	EQ	.398836E 01	.399363E 01	.527000E-02	.378641E 01

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
CONSTRAINT UPBO	LT	-.999262E 00	-.100000E 01	-.737548E-03	-.125494E 02

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
** CONSTRAINT NL4	LT	.812564E 01	.105170E 02	.239135E 01	.000000E 00
** CONSTRAINT NL5	LT	-.594478E 01	-.447377E 01	.147101E 01	.000000E 00
** CONSTRAINT NL6	LT	-.614618E 01	-.542500E 01	.721188E 00	.000000E 00
CONSTRAINT LIN1	GT	.385594E 01	.200000E 01	-.185594E 01	.000000E 00
CONSTRAINT LIN2	LT	-.299499E 01	-.200000E 01	.994986E 00	.000000E 00
** CONSTRAINT NL3	LT	.884765E 01	.139215E 02	.507390E 01	.000000E 00
CONSTRAINT LOBO	GT	.185912E 01	.100000E 01	-.859124E 00	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.520325E-02	NL2 =	.145146E 01	NL3 =	-.104742E 01
NL4 =	-.574230E 00	NL5 =	-.248362E 01	NL6 =	-.714412E 00

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

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

THE LINEAR CONSTRAINTS

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

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
** CONSTRAINT NL2	EQ	.744975E 01	.744979E 01	.000000E 00	-.672945E 00
** CONSTRAINT NL1	EQ	.399763E 01	.399768E 01	.000000E 00	.534709E 00

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
** CONSTRAINT NL6	LT	-.671326E 01	-.671334E 01	.000000E 00	-.144268E 00
CONSTRAINT UPB0	LT	-.999993E 00	-.100000E 01	.000000E 00	-.259913E 00

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
** CONSTRAINT NL5	LT	-.998117E 01	-.748141E 01	.249976E 01	.000000E 00
CONSTRAINT LIN1	GT	.345765E 01	.200000E 01	-.145765E 01	.000000E 00
CONSTRAINT LIN2	LT	-.302096E 01	-.200000E 01	.102096E 01	.000000E 00
** CONSTRAINT NL4	LT	.153328E 02	.198513E 02	.451848E 01	.000000E 00
CONSTRAINT LOB0	GT	.147872E 01	.100000E 01	-.478724E 00	.000000E 00
**CONSTRAINT NL3	LT	.269201E 02	.278998E 02	.979660E 00	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.696182E-04	NL2 =	.144930E 00	NL3 =	-.979677E 00
NL4 =	-.376660E 01	NL5 =	-.249976E 01	NL6 =	.537645E-03

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

	VARIABLE ACTIVITY	COMPUTED GRADIENT	GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS
VARIABLE X(1)	.145861E 01	.291695E 00	.315945E 00
VARIABLE X(2)	.197768E 01	.395506E 01	.399333E 01
VARIABLE X(3)	-.100515E 01	-.201065E 02	-.200731E 02
VARIABLE X(4)	-.296368E-01	-.592737E 01	-.588883E 01

THE LINEAR CONSTRAINTS

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

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL2	EQ	.615883E 01	.614393E 01	-.148954E-01	-.116902E 01
**CONSTRAINT NL1	EQ	.398744E 01	.399942E 01	.119762E-01	.614307E 01

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL6	LT	-.603298E 01	-.604150E 01	-.851727E-02	-.185339E 01

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL4	LT	.956768E 01	.134662E 02	.389855E 01	.000000E 00
**CONSTRAINT NL5	LT	-.991945E 01	-.749838E 01	.242107E 01	.000000E 00
CONSTRAINT LIN1	GT	.340665E 01	.200000E 01	-.140665E 01	.000000E 00
CONSTRAINT LIN2	LT	-.301247E 01	-.200000E 01	.101247E 01	.000000E 00
CONSTRAINT UPBO	LT	-.100515E 01	-.100000E 01	.515175E-02	.000000E 00
CONSTRAINT LOBO	GT	.145861E 01	.100000E 01	-.458609E 00	.000000E 00
**CONSTRAINT NL3	LT	.267714E 02	.279940E 02	.122255E 01	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.119896E-01	NL2 =	.165850E-01	NL3 =	-.121968E 01
NL4 =	-.389678E 01	NL5 =	-.242155E 01	NL6 =	.884059E-02

LINEARLY CONSTRAINED OPTIMUM FOUND AFTER 5 MINOR ITERATIONS.
TOTAL NUMBER OF MINOR ITERATIONS UNTILL NOW: 92

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

THE VARIABLES

	VARIABLE ACTIVITY	COMPUTED GRADIENT	GRADIENT, CALCULATED FROM LAGRANGE MULTIPLIERS AND CONSTRAINT NORMALS
VARIABLE X(1)	.145521E 01	.290933E 00	.288354E 00
VARIABLE X(2)	.197941E 01	.395869E 01	.395460E 01
VARIABLE X(3)	-.101013E 01	-.202037E 02	-.202349E 02
VARIABLE X(4)	-.298259E-01	-.596518E 01	-.599194E 01

THE LINEAR CONSTRAINTS

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

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL2	EQ	.601591E 01	.601569E 01	.000000E 00	-.119905E 01
**CONSTRAINT NL1	EQ	.401045E 01	.401045E 01	.210762E-03	.615135E 01

THE ACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL6	LT	-.605054E 01	-.605094E 01	-.405312E-03	-.187001E 01

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL4	LT	.928767E 01	.132061E 02	.391841E 01	.000000E 00
**CONSTRAINT NL5	LT	-.985973E 01	-.742130E 01	.243843E 01	.000000E 00
**CONSTRAINT NL3	LT	.263157E 02	.274996E 02	.118394E 01	.000000E 00
CONSTRAINT LIN1	GT	.340480E 01	.200000E 01	-.140480E 01	.000000E 00
CONSTRAINT LIN2	LT	-.301937E 01	-.200000E 01	.101937E 01	.000000E 00
CONSTRAINT UPBO	LT	-.101013E 01	-.100000E 01	.101318E-01	.000000E 00
CONSTRAINT LOBO	GT	.145521E 01	.100000E 01	-.455214E 00	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.222206E-03	NL2 =	.220597E-03	NL3 =	-.118399E 01
NL4 =	-.391840E 01	NL5 =	-.243844E 01	NL6 =	.449453E-03

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

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

THE LINEAR CONSTRAINTS

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

THE INDEPENDENT EQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL2	EQ	.599899E 01	.599906E 01	.000000E 00	-.119517E 01
**CONSTRAINT NL1	EQ	.402033E 01	.402037E 01	.000000E 00	.614748E 01

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL6	LT	-.605918E 01	-.605929E 01	.000000E 00	-.186542E 01

THE INACTIVE INEQUALITIES

	TYPE	ACTIVITY	RIGHT HAND SIDE	SLACK ACTIVITY	LAGRANGE MULTIPLIER
**CONSTRAINT NL4	LT	.924338E 01	.131618E 02	.391839E 01	.000000E 00
**CONSTRAINT NL5	LT	-.987645E 01	-.743795E 01	.243850E 01	.000000E 00
CONSTRAINT LIN1	GT	.340473E 01	.200000E 01	-.140473E 01	.000000E 00
CONSTRAINT LIN2	LT	-.301945E 01	-.200000E 01	.101945E 01	.000000E 00
CONSTRAINT UPBO	LT	-.101023E 01	-.100000E 01	.102262E - 01	.000000E 00
CONSTRAINT LOBO	GT	.145521E 01	.100000E 01	-.455209E 00	.000000E 00
**CONSTRAINT NL3	LT	.263876E 02	.275716E 02	.118401E 01	.000000E 00

VALUES OF THE NONLINEAR CONSTRAINTS ARE:

CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY	CONSTRAINT NAME	ACTIVITY
NL1 =	-.696182E-04	NL2 =	-.975132E-04	NL3 =	-.118412E 01
NL4 =	-.391843E 01	NL5 =	-.243848E 01	NL6 =	.121638E-03

LIST OF REPORTS 1978

7800 "List of Reprints, nos 200-208; Abstracts of Reports Second Half 1977".

7801/S "Conjugate TT-Variation and Process Inversion", by L. de Haan and S.J. Resnick.

7802/S "General Quadratic Forms in Normal Variates", by C. Dubbelman.

7803/S "On Bahadur's Representation of Sample Quantiles", by L. de Haan and E. Taconis-Haantjes.

7804/O "Experiments with a Reduction Method for Nonlinear Programming Based on a Restricted Lagrangian", by G. v.d. Hoek.

7805/S "Derivatives of Regularly Varying Functions in IR^d and Domains of Attraction of Stable Distributions", by L. de Haan and S.I. Resnick.

7806/E "Estimation and Testing of Alternative Production Function Models", by S. Schim van der Loeff and R. Harkema.

7807/E "Money Illusion and Aggregation Bias", by J. van Daal.

7808/M "Aspects of Elliptic Curves: An Introduction", by R.J. Stroeker.

7809/E "Analytical Utility Functions Underlying Fractional Expenditure Allocation Models", by W.H. Somermeyer and J. van Daal.

7810/M "A New Proof of Cartier's Third Theorem", by M. Hazewinkel.

7811/M "On the (Internal) Symmetry Groups of Linear Dynamical Systems", by M. Hazewinkel.

7812/E "Empirical Evidence on Pareto-Lévy and Log Stable Income Distributions", by H.K . van Dijk and T. Kloek.

7813/E "A Family of Improved Ordinary Ridge Estimators", by A. Ullah, H.D. Vinod and R.K. Kadiyala.

7814/E "An Improvement of the Main Program from Report 7304", by C. Dubbelman.

7815 "Publications of the Econometric Institute, First Half 1978: List of Reprints, Nos. 209-219; Abstracts of Reports.

7816/O "A Hierarchical Clustering Scheme for Asymmetric Matrices", by D.S. Brée, B.J. Lageweg, J.K. Lenstra, A.H.G. Rinnooy Kan and G. van Bezouwen.

7817/O "Generating All Maximal Independent Sets: NP-Hardness and Polynomial-Time Algorithms", by E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan.

7818/M "Note on the Eigenvalues of the Covariance Matrix of Disturbances in the General Linear Model", by R.J. Stroeker.

7819/S "An Extension of Karamata's Tauberian Theorem and its Connection with Complementary Convex Functions", by A.A. Balkema, J.L. Geluk and L. de Haan.

7820/S "An Abel-Tauber Theorem on Convolutions with the Möbius Functions", by J. Geluk.

7819/S "An Extension of Karamata's Tauberian Theorem and its Connection with Complementary Convex Functions", by A.A. Balkema, J.L. Geluk and L. de Haan.

7820/S "An Abel-Tauber Theorem on Convolutions with the Möbius Functions", by J. Geluk.

7821/O "Optimization Methods Based on Projected Variable Metric Search Directions", by J.F. Ballintijn, G. van der Hoek and C.L. Hooykaas.