OPTIMIZATION IN LARGE PARTLY NONLINEAR SYSTEMS

Arne Drud
IMSOR
The Institute of Mathematical Statistics
and Operations Research
The Technical University of Denmark
DK 2800 Lyngby - Denmark

0. Abstract.

This paper describes a method for optimizing large partly nonlinear systems. The method is based on the GRG-algorithm, that solves problems with nonlinear objective function and nonlinear equality constraints. The original GRG-algorithm is described and its relations with LP are stressed. Some storage problems in large problems are discussed, and a special inversion procedure for the GRG-algorithm is presented. Some special kinds of constraints, inequalities and linear constraints, are considered, and it is shown, how their special features can be utilized. Finally some computational results with the method are given.

1. Introduction.

The purpose of the research project, that is described in this paper, was to develop an optimization procedure, that could be used in connection with large econometric models.

Econometric models have some features, that are important for the way the optimization procedure is designed. First of all, most constraints in the model are equality constraints. Many equations are linear, but there are usually also some nonlinear equations. There can be lower and/or upper bounds on many variables. And finally there are only a few active variables in each constraint.

The GRG-algorithm by J. Abadie, [1], [2], and [3], is made for problems of this kind, and a first approach was therefore to use the GRG69 computer program, that ranges very high on the list of Colville, [5]. But we very soon ran into problems, because the program needed too much core storage for arrays.

This paper describes, how a GRG-type program can be designed, taking into account the sparseness of the Jacobian of the constraints. The program uses less core storage than the GRG69 program, and due to a fast inversion procedure, it is also faster.

In section 2 the ideas in the GRG-algorithm are described, and some important subproblems are described in section 3 and 4. Section 5 explains how the Jacobian can be stored, and section 6 contains an inversion procedure specially designed for the GRG-algorithm. Section 7 is on the special treatment of inequalities and linear equations, and section 8 shows some computational results. The conclusion is in section 9.

2. The Generalized Reduced Gradient Method.

In this section the main ideas in the GRG-method will be explained for easy reference in the rest of the paper.

Consider the problem:

$$\max \quad z = f(\underline{x}) \tag{1}$$

$$\text{subject to} \quad \underline{g}(\underline{x}) = \underline{0} \tag{2}$$

$$\text{and} \quad \underline{\alpha} \leq \underline{x} \leq \underline{\beta} \tag{3}$$

where $\underline{x}$, $\underline{\alpha}$, and $\underline{\beta}$ are m-vectors and $\underline{g}$ is an n-dimensional vectorfunction, ($n < m$). The functions f and $\underline{g}$ are assumed to be at least one time differentiable with known continuous partial derivatives.

A set of n equalities like (2) can be used to eliminate n variables. Partition the $\underline{x}$-vector into $\underline{x}_b$ and $\underline{x}_n$, where $\underline{x}_b$ has n elements and $\underline{x}_n$ has m-n elements, and transform (2) into $\underline{x}_b = \underline{g}_1(\underline{x}_n)$. Maybe it is not possible to find an analytic expression for $\underline{g}_1$, but from the theory of implicit functions we have the following theorem: If $(\underline{x}_b^o, \underline{x}_n^o)$ satiesfies $\underline{g}(\underline{x}_b^o, \underline{x}_n^o) = \underline{0}$ and the Jacobian $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ has rank n, i.e. is nonsingular, then in a neighbourhood of $(\underline{x}_b^o, \underline{x}_n^o)$ it is possible to transform $\underline{g}(\underline{x}_b, \underline{x}_n) = \underline{0}$ into $\underline{x}_b = \underline{g}_1(\underline{x}_n)$. The function $\underline{g}_1$ is differentiable, and the Jacobian is found by implicit differentiation:

$$\frac{\partial \underline{g}}{\partial \underline{x}_b} \cdot \frac{\partial \underline{x}_b}{\partial \underline{x}_n} + \frac{\partial \underline{g}}{\partial \underline{x}_n} = \underline{0}$$

$$\frac{\partial \underline{g}_1}{\partial \underline{x}_n} = \frac{\partial \underline{x}_b}{\partial \underline{x}_n} = -\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1} \cdot \frac{\partial \underline{g}}{\partial \underline{x}_n} \tag{4}$$

The equation $\underline{x}_b = \underline{g}_1(\underline{x}_n)$ is now introduced in the objective function:

$$z = f(\underline{x}_b, \underline{x}_n) = f(\underline{g}_1(\underline{x}_n), \underline{x}_n) = F(\underline{x}_n)$$

where $F(\underline{x}_n)$ is differentiable with derivative:

$$\frac{\partial F}{\partial \underline{x}_n} = \frac{\partial f}{\partial \underline{x}_b} \cdot \frac{\partial \underline{x}_b}{\partial \underline{x}_n} + \frac{\partial f}{\partial \underline{x}_n} = \frac{\partial f}{\partial \underline{x}_n} - \left(\frac{\partial f}{\partial \underline{x}_b} \cdot \left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}\right) \cdot \frac{\partial \underline{g}}{\partial \underline{x}_n} \qquad (5)$$

The problem (1), (2), and (3) can now be reformulated:

$$\text{max} \qquad z = F(\underline{x}_n) \qquad (6)$$

$$\text{subject to} \qquad \underline{\alpha}_n \leq \underline{x}_n \leq \underline{\beta}_n \qquad (7)$$

$$\text{and} \qquad \underline{\alpha}_b \leq \underline{x}_b \leq \underline{\beta}_b \qquad (8)$$

$$\text{where} \qquad \underline{x}_b = \underline{g}_1(\underline{x}_n) \qquad (9)$$

The transformations done until now are very similar to those done in an LP-problem. $\underline{x}_b$ is the set of basic variables. They are introduced to compensate for changes in the non-basic $\underline{x}_n$-variables, so that (2) will still hold. The derivatives $\frac{\partial F}{\partial \underline{x}_n}$ are similar to the reduced costs in an LP-problem. They measure the influence on the objective function of changes in the non-basic variables, taking into account the corresponding changes in the basic variables. $\frac{\partial F}{\partial \underline{x}_n}$ is called the reduced gradient. The matrix $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ plays the same role as the basic-matrix does in LP.

A major difference between this problem and an LP-problem is, that in the optimal solution this problem can have more than n variables between bounds. Thus, it is not possible to use an optimization procedure like the simplex, that only works with basic solutions.

The major steps in the GRG-algorithm are now as shown in fig. 1. In 2) the basic variables are chosen strictly between the bounds, so that they can actually compensate for changes in $\underline{x}_n$ without exceeding a bound at once. The vector $\underline{u}$ in 4) is similar to the simplex multipliers in LP, and $\underline{rg}$ is the reduced gradient found in (5). $\underline{h}$ is the reduced gradient projected on the simple inequality constraints, and if it is zero, the Kuhn-Tucker conditions are satiesfied. While choosing the optimal θ in 8) it can be valuable to notice, that $\frac{\partial z}{\partial \theta}\big|_{\theta=0} = \underline{h}^T \cdot \underline{rg}$. Step 8b) will be described in next section.

The algorithm in fig. 1 uses the steepest ascend method. If the nonbasic variables are the same from iteration to iteration, we are optimizing $F(\underline{x}_n)$ with the same $\underline{x}_n$-variables, and therefore it is possible to use

1) Find a first feasible solution, $\underline{x}^{o}$.

2) Calculate $\frac{\partial \underline{g}}{\partial \underline{x}}$ and choose n variables among the $\underline{x}$-variables, $\underline{x}_b$, so that $\underline{\alpha}_b < \underline{x}_b^{o} < \underline{\beta}_b$ with a strict inequality, and so that $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ is nonsingular. Name the rest of the variables $\underline{x}_n$.

3) Find the inverse of $\frac{\partial \underline{g}}{\partial \underline{x}_b}$, $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$.

4) Compute $\underline{u}^T = -\left(\frac{\partial f}{\partial \underline{x}_b}\right)^T \cdot \left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$.

5) Compute $\underline{rg} = \frac{\partial f}{\partial \underline{x}_n} + \underline{u}^T \cdot \frac{\partial \underline{g}}{\partial \underline{x}_n}$

6) Find $\underline{h}$ as: $h_i = \begin{cases} 0 \text{ if } rg_i < 0 \text{ and } x_{ni}^{o} = \alpha_{ni} \\ 0 \text{ if } rg_i > 0 \text{ and } x_{ni}^{o} = \beta_{ni} \\ rg_i \text{ else} \end{cases}$

7) If $\underline{h} = \underline{0}$, then stop.

8) Choose θ to maximize $F(\underline{x}_n^{o} + \theta \cdot \underline{h})$ by solving a)-c) for different θ-values.

a) $x_{ni} = \begin{cases} \alpha_{ni} \text{ if } x_{ni}^{o} + \theta \cdot h_i < \alpha_{ni} \\ \beta_{ni} \text{ if } x_{ni}^{o} + \theta \cdot h_i > \beta_{ni} \\ x_{ni}^{o} + \theta \cdot h_i \text{ else} \end{cases}$

b) Find $\underline{x}_b = \underline{g}_1(\underline{x}_n)$

c) $z = f(\underline{x}_b, \underline{x}_n)$

9) Store the best solution until now in $\underline{x}^{o}$ and go to 2.

Figur 1. The GRG-algorithm.

a conjugate gradient method. Fletcher-Reeves method [6] is probably the best method, because it requires a very limited amount of core storage.

3. The implicit function $\underline{x}_b = \underline{g}_1(\underline{x}_n)$.

A very important problem in optimization problems with equality constraints is to stay on the surface defined by $\underline{g}(\underline{x}) = \underline{0}$, or in the words of the GRG-algorithm: Find $\underline{x}_b = \underline{g}_1(\underline{x}_n)$.

The function $\underline{g}_1$ is not known explicitly, so $\underline{x}_b$ must be found as a solution to the set of equations $\underline{g}(\underline{x}_b, \underline{x}_n) = \underline{0}$, where $\underline{x}_n$ is known, i.e. n nonlinear equations must be solved for n unknown.

The classical method is the Newton-Raphson method:

$$\underline{g}(\underline{x}_b+\Delta\underline{x}_b,\underline{x}_n) \approx \underline{g}(\underline{x}_b,\underline{x}_n) + \frac{\partial \underline{g}}{\partial \underline{x}_b}\cdot\Delta\underline{x}_b = \underline{0}$$

$$\Delta\underline{x}_b = -\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}\cdot\underline{g}(\underline{x}_b,\underline{x}_n)$$

$$\underline{x}_b := \underline{x}_b + \Delta\underline{x}_b$$

It is an iterative method, and with a good starting point it is very fast. As a stop criterion we can use

$$\sum_{i=1}^{n}\left(g_i(\underline{x}_b,\underline{x}_n)\right)^2 < \varepsilon_{new}$$

or any other convenient measure of precision.

It can be very timeconsuming to calculate $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ and invert it in each new point. But $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ is continuous, so for small steplengths the matrix calculated in step 2 of the GRG-algorithm and the inverse from step 3 can be used. When a constant Jacobian is used, the method is sometimes called the pseudo Newton-Raphson method. The use of the same Jacobi-matrix and its inverse both in step 4 of the GRG-algorithm and to compute the implicit function $\underline{g}_1$ is one of the main advantages of the GRG-algorithm.

If the steplength θ in step 8 is too large, $\underline{g}(\underline{x}_b,\underline{x}_n)$ can be far from $\underline{0}$, and the inverse Jacobian from step 2 may be far from the inverse Jacobian in the actual point, so the pseudo Newton-Raphson method does not converge. The best treatment of this problem is probably to stop after a fixed number of iterations and then decrease the value of θ.

4. Lower and upper bounds on the basic variables.

Until now only the bounds on the nonbasic variables have been treated. When the set of nonlinear equations $\underline{g}(\underline{x}_b,\underline{x}_n) = \underline{0}$ is solved with respect to $\underline{x}_b$, the guess for $\underline{x}_b$ is changed in each pseudo Newton-Raphson iteration. What can be done, if one or more basic variables exceed a bound? The problem is basically the same that occure in LP, when a basic variable becomes too small or too large.

The situation is: $\underline{x}_b$ lies within the bounds and $\underline{x}_b+\Delta\underline{x}_b$ lies outside. Choose the largest α so that $\underline{x}_b+\alpha\Delta\underline{x}_b$ lies inside or on the bounds, and

replace $\underline{x}_b$ by $\underline{x}_b + \alpha \cdot \Delta\underline{x}_b$. This operation should decrease the error in $\underline{g}(\underline{x}_b, \underline{x}_n) = \underline{0}$, but not as much as for $\alpha = 1$. Now one of the basic variables is at a bound, and it cannot compensate for changes in $\underline{x}_n$. Therefore perform a change of basis: Choose an $\underline{x}_n$-variable, that lies strictly between the bounds and introduce it as a basic variable, while the one at bound is transfered to $\underline{x}_n$. The only thing we need in order to proceed with the pseudo Newton-Raphson method is the inverse Jacobian $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$ with the new basic variables $\underline{x}_b$.

Only one column in $\frac{\partial \underline{g}}{\partial \underline{x}_b}$ is exchanged with another. This is exactly what happens in a change of basis in the simplex procedure, and the procedure for updating $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$ is the same. The actual calculations depend on the way $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$ is represented and will be described after the section on inversion procedures.

5. Storing the Jacobian.

Large problems means in this paper problems with more than 250 constraints and more than 300 variables. Therefore the Jacobian will be a matrix with more than 75,000 elements, and even for a large modern computer, this is a large matrix.

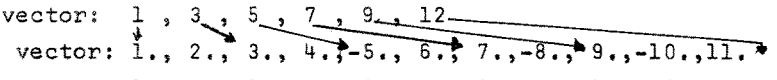
However in most reallife problems only a few variables are active in each equation. This means, that may be only 1000 - 2000 out of the possible 75,000 elements in the Jacobian are nonzero. So if only the nonzero elements are stored with a reference to the place in the matrix, a lot of core storage can be saved.

In the different steps of the GRG-algorithm we always use one column of the Jacobian at a time. So it is very efficient to store the elements column by column in a long vector with another integer-valued vector telling the row-number of each element, and a shorter vector telling where in the long vectors the different columns start.

Example:

$$\frac{\partial \underline{g}}{\partial \underline{x}} = \begin{Bmatrix} 1. & 0. & 0. & 0. & 9. \\ 0. & 3. & -5. & 7. & 0. \\ 0. & 4. & 6. & 0. & -10. \\ 2. & 0. & 0. & -8. & 11. \end{Bmatrix}$$

Storage pattern:



6. Inversion procedures.

Step 3 of the GRG-algorithm in fig. 1 read:"find the inverse of $\frac{\partial \underline{g}}{\partial \underline{x}_b}$, $(\frac{\partial \underline{g}}{\partial \underline{x}_b})^{-1}$."

We do not need the inverse, which can easily be a 62,500 element matrix, in explicit form. Some representation of the inverse, that makes it possible to multiply a vector from the right or from the left with the inverse, is sufficient.

Fortunately these problems have been worked on for a long time in connection with linear programming, where the two matrix multiplications are used to find the simplix multipliers and to find a transformed column. Some of the techniques have been described in [4], [8], [9], [10], and [11].

Section 6.1 will describe the idea of representing the inverse as a string of eta-vectors, and section 6.2 describes an inversion procedure, that performs the inversion and the selection of the basic variables simultaneously.

6.1 The inverse on product form.

According to the definition of an inverse matrix, the product $\underline{x} = \underline{A}^{-1} \cdot \underline{y}$ is the solution to the set of equations $\underline{A} \cdot \underline{x} = \underline{y}$. A set of linear equations is usually solved by performing a set of row operations on $\underline{A}$ and $\underline{y}$. When $\underline{A}$ has been transformed into the unit matrix $\underline{I}$, then $\underline{y}$ has been transformed into $\underline{A}^{-1} \cdot \underline{y} = \underline{x}$.

Consider the following row operations: multiply row i by $1/a_{ii}$, multiply row i by $-a_{ji}/a_{ii}$ and add it to row j, $j = 1,\ldots,n$, $j \neq i$. The rowoperations are equivalent to a multiplication from the left with the matrix $\underline{E}_i$:

$$\underline{\underline{E}}_i = \begin{bmatrix} 1 & 0 & \ldots & -a_{1i}/a_{ii} & \ldots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \ldots & 1/a_{ii} & \ldots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \ldots & -a_{ni}/a_{ii} & \ldots & 0 \end{bmatrix}$$

Now the inverse can be represented as a sequence of $\underline{\underline{E}}$-matrices, each specifying some of the row operations. The $\underline{\underline{E}}$-matrices are usually called eta-vectors - vectors, because they only differ from the unit matrix in one vector.

The matrix $\underline{\underline{E}}_i$ can be stored very compact. First of all only the column $a_{.i}$ should be stored, and in this column it is enough to store the non-zero elements. As shown in section 6.2 it is possible to keep the number of nonzero elements low by choosing the pivot elements a_{ii} carefully. And with fewer eta-elements less core storage should be used, and fewer row operations must be performed during the use of the inverse.

6.2 The GRG-inversion procedure.

There is a main difference between the linear programming inversion procedure and the procedure, that is used in the GRG-problem. In the later there is an extra degree of freedom, because there are usually more variables, that can be choosen as basic variables, than needed.

The procedure described here is heavily based on the procedure for LP-problems described by Hellerman and Rarick, [8] and [9].

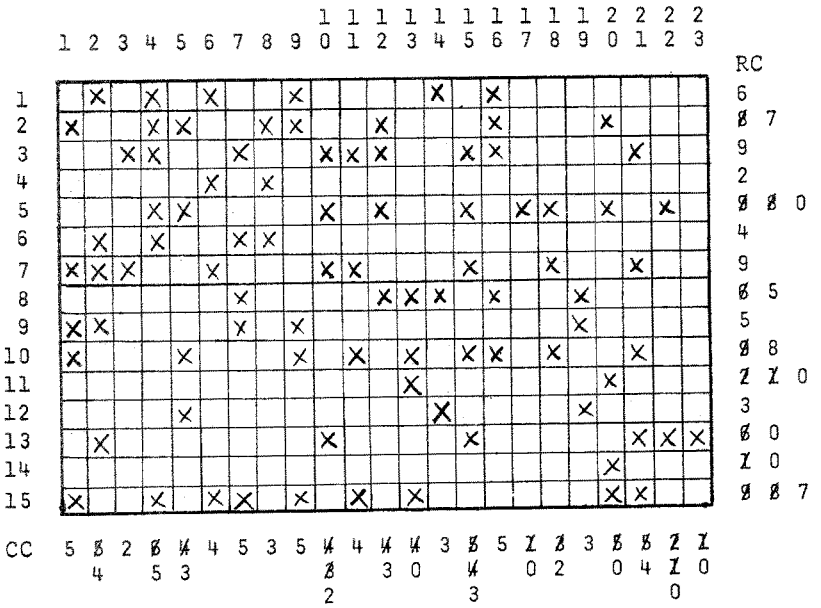
First a basic observation must be made. The first eta-vector is created very easily - it is created directly from the column in which we pivot first. The second eta-vector must be chosen from the matrix after the row operations symbolized by $\underline{\underline{E}}_1$ have been performed. But if the column to be used for pivot next is chosen, so it has no element in the first pivotrow, the row operations does not effect this column.

It is seen, that if the matrix can be transformed into lower-triangular form by rearranging rows and columns, and if the pivotelements are chosen along the diagonal from the upper left corner, no columns will have to be transformed before the eta-vectors are created.

Usually it is not possible to transform the matrix into lower-triangular

form. The Hellerman-Rarick procedure tries systematically to create a lower-triangular matrix, and if it is not possible, the procedure selects the columns to be transformed (the so-called spikes) in such a way, that the rest of the matrix becomes almost triangular.

The procedure in its form for the GRG-problem will be described with a small illustrative example. In the following, only the columns strictly between bounds are used. Fig. 2 shows the zero-nonzero pattern of the matrix.



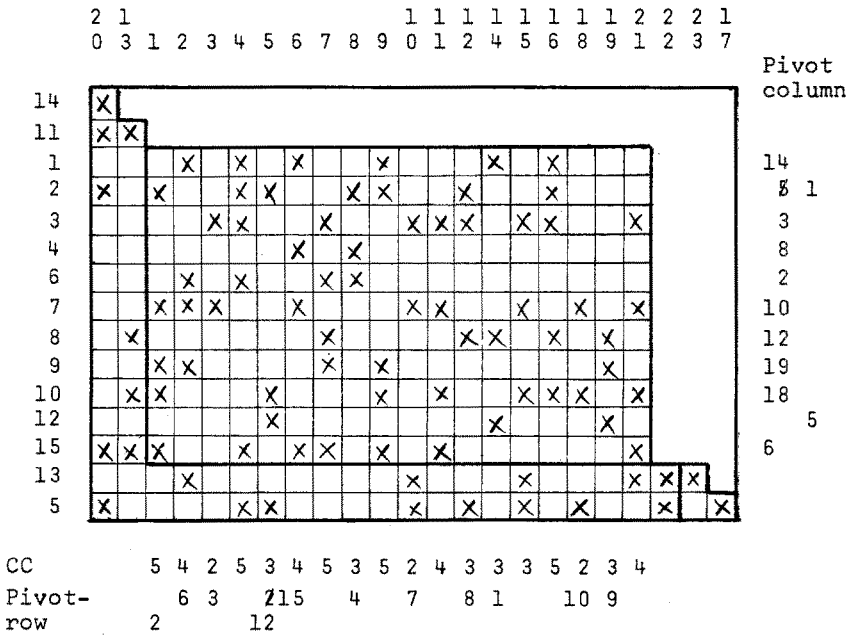
Figur 2: The zero-nonzero pattern of the matrix.

The first steps identifies the part of the matrix, that directly is lower-triangular:

1. Find the rowcounts (RC) and columncounts(CC), i.e. the number of non-zero elements in each row and column.
2. Is there an $RC_i = 1$? If not, go to 3.
   Find the corresponding column, choose the element as pivot-element, delete the column and revise RC and CC, and go to 2.
3. If all rows have a pivotelement, then go to 17.
4. Is there an $CC_j = 1$? If not, go to 5.
   a. If there are more columns with $CC_j = 1$, choose one with the smallest number of nonzero elements.

b. Find the corresponding row, choose the element as pivotelement, delete the row and revise RC and CC and go to 4.

5. If all rows have a pivotelement, then go to 17.

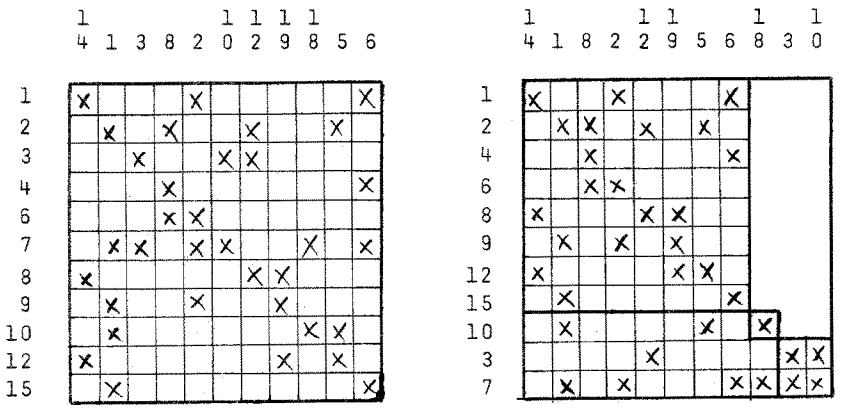


Figur 3: The matrix after step 1 to 5.

After rearranging the matrix, it is now divided into three parts: two lower-triangular parts and a part with either no elements or at least two elements pr row and pr column. See fig. 3. From now on only the part in the middle is considered.

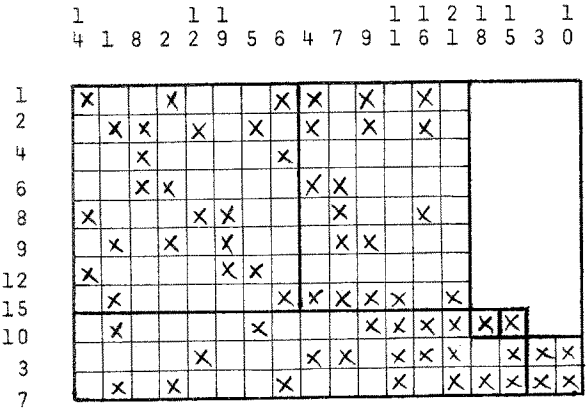
6. Perform a temporary pivotassignment:
   a. Consider all columns starting with the columns with the smallest CC_j. Use the first nonzero element found in a row, that has no pivotelement yet, as pivot element. If a column cannot be assigned a pivot element after this procedure, the next column is considered.
   b. If all rows have a pivot element, then go to 7.
   c. Use the Ford and Fulkerson labelling method to reassign pivot elements, until we have a maximum matrix transversal, see [7].
7. Now consider the matrix of those columns, that were assigned pivot elements in step 6. Decompose this matrix into minimal nondecomposable submatrices, f.ex. with the predecessor-successor method in Hellerman and Rarick [9], or with the method of Steward [12].

Step 6 and 7 used on the example results in the situation in fig. 4. After the assignment in step 6a, row 12 is still without a pivot element, which is found by the labeling procedure.



Figur 4: The matrix after step 6 and 7.

8. Decompose the middle part of the matrix in the subblocks corresponding to the decomposition in submatrices in 7, se fig. 4 and 5.



Figur 5: The subblocks formed in step 8.

The following steps 9 to 16 are now done for each subblock, starting with the upper left corner. The temporary pivotassignments are not used any more.

9. Find RC and CC for this subblock.
10. Perform the operations in step 4 on the subblock.

The result is now a matrix with at least two elements per row and per column, so it is not possible to get a lower triangular matrix directly. The procedure is now to select the columns, that should be transformed or that should be left out completely. These columns are called spikes. For the selection of spikes, Hellerman and Rarick defined a tallyfunction in the following way:

$t_k(j)$ = number of nonzero elements in column j, appearing in rows with RC $\leq$ k.

The selection of a spike is now done with the following procedure, where k starts as the smallest RC:

a. S = The set of columns for which $t_k(j)$ is maximum.
b. If S has only one element, then return.
c. If $t_{max} > 1$, then go to g.
d. k:= the smallest RC > k.
e. S:= the subset of S for which $t_k(j)$ is maximum.
f. Go to b
g. S:= the subset of S for which CC is maximum.
h. If S has only one element, then return.
i. S:= the subset of S for which the total column count is maximum.
j. Return with any element from S as the spike.

The ideas behind the selection procedure are:

a. Make the smallest rowcount smaller, so we can get a row with $RC_i = 1$.
b. If there are more possibilities, choose the one that will bring most small rowcounts down.
c. If there are still more possibilities, choose the column with the largest number of elements.

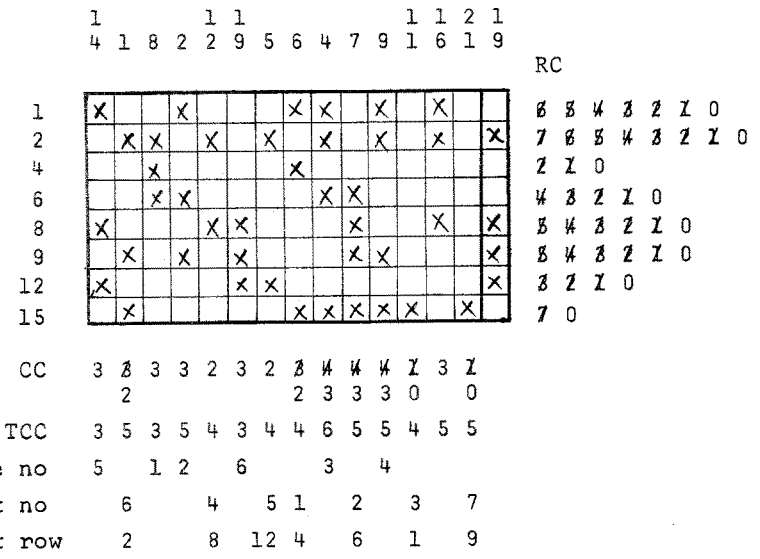
Now the pivot selection procedure can be continued:

11. k:= the smallest RC in rows without a pivot element.
12. If k = 0, go to 16, if k = 1, go to 14, else go to 13.
13. Select a spike, save its number, delete the column, revise RC and CC, and go to 11.
14. If there are more $RC_i = 1$, then go to 15, else select row no i and the corresponding column for final pivot, delete the column, revise RC and CC, and go to 11.
15. Select the pivot column. The procedure for selecting spikes is used with point i exchanged by
    i1. S:= the subset of S for which the total columncount is minimum.
    Choose the pivot row as one with $RC_i = 1$, delete the column, revise

RC and CC, and go to 11.

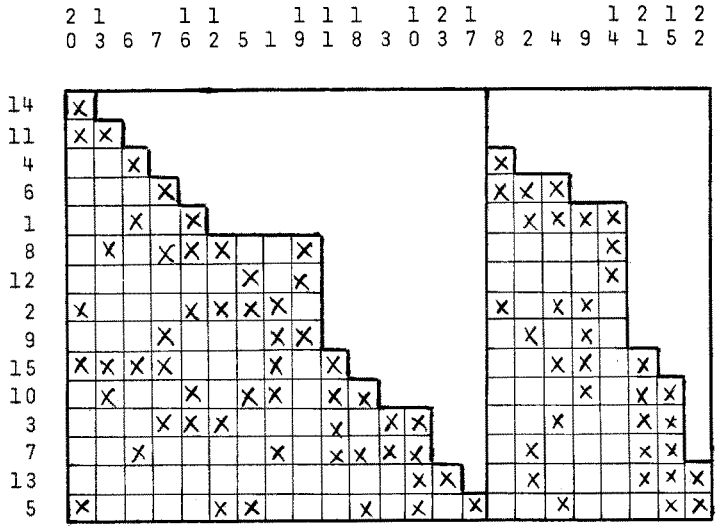
16. The last column in the list of spikes is transformed with the eta-vectors, and the resulting column is transformed into an eta-vector with pivot in a row with $RC_i = 0$ and without a pivot element. If all rows in the block have a pivot element now, this block has been finished, else go to 11.

Fig. 6 shows how the procedure works on the largest subblock. Fig. 7 shows the matrix after all rearrangements have been done, and fig. 8 shows the corresponding eta-vectors. Some columns has been divided into two eta-vectors, so that the transformation of the spike will only create new elements in the upper part of the spike. This idea is described by Beale in [4].

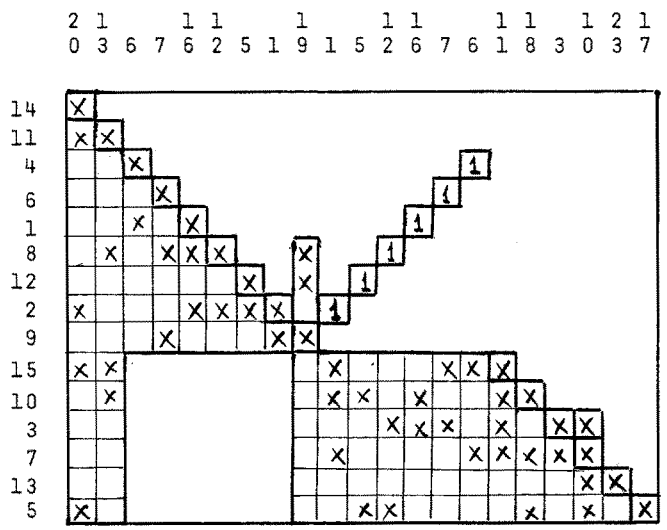


Figur 6: The final pivot selection in a subblock. RC and CC are row- and columncounts, TCC is columncounts in the original matrix, Spike no and Pivot no indicates, when a column has been chosen for either spike or pivot, Pivot row tells, in which row the pivot of the column can be found.

Sometimes a pivotelement will be found to be too small. In this case it is relatively easy to delete the corresponding column, because it is linearly dependent on earlier columns, and use the last spike-column instead.



Figur 7: The matrix after the final rearrangements.



Figur 8: The resulting eta-vectors.

6.3 Updating the inverse after changes of basis.

In section 4 we saw, that a change of basis can occure in a Newton-Raphson iteration, and that the operations needed to update the inverse are exactly the same as those of the simplex method, i.e. transform the incomming column and use this column as an extra eta-vector with pivot in

the same row as the column, that is leaving.

This method has a great advantage. In the GRG-algorithm it is often nessecary to try with different steplengths, θ, before the optimal steplength in the direction of the reduced gradient is found. Each time a new θ-value is tried, we need the $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1}$ matrix, where $\underline{x}_b$ is the vector of original basic variables. If the last θ-value caused any changes of basis, it is now nessecary to restore the old inverse basis matrix. This is easily done with the inverse on product form: just delete the extra eta-vectors, that were added during the changes of basis.

7. Inequalities and linear equations.

The GRG-method is basically designed for nonlinear equality constraints. This section shows, what happens with inequalities and with linear equations.

7.1 Inequalities.

It is always possible to change an inequality into an equality constraint by adding a slack variable:

$$g_i(\underline{x}) \geq 0$$

$$g_i(\underline{x}) - s_i = 0, \quad 0 \leq s_i < \infty$$

There are now two cases to consider:

The constraint is not active, i.e. $s_i > 0$. Now the combined inversion and basis-selection procedure will select s_i as a basic variable during step 4, and the eta-vector will only have one nonzero element, the pivot element -1. It is easy to show, that the $\underline{u}$-vector in the GRG-algorithm will have a zero in position i, which means, that row no i has no influence on the reduced gradient. And in the Newton-Raphson iterations, equation no i will converge very fast, because s_i will just be assigned the value $g_i(\underline{x})$.

The constraint is active, i.e. $s_i = 0$. Now s_i cannot be used as a basic variable, and almost all calculations will be as if the equation had been $g_i(\underline{x}) = 0$.

It can be seen, that the practical calculations are very similar to those

of a relaxation procedure, where only the active constraints are used when deciding where to go in next iteration.

7.2 Linear equations.

Although a model is nonlinear, it will usually have many equality constraints, such as continuity constraints, equations found by linear regression etc. Linear equations are mathematically much simpler than nonlinear ones, and it is indeed possible to take advantage of this simplicity.

Linear equations will have zero error after the first Newton-Raphson iteration, because the linear approximation used in the Newton-Raphson procedure is identical to the linear equation itself. Therefore it is not nessecary to compute the value of the $\underline{g}$-vector for the linear equations after the first Newton-Raphson iteration. This again will make the calculation of $\left(\frac{\partial \underline{g}}{\partial \underline{x}_b}\right)^{-1} \cdot \underline{g}$ faster, since it is not nessecary to perform the multiplications, where one of the factors is zero.

8. Computational experience.

8.1 User routines.

The computer program, that has been developed, uses four problem defining subroutines.

The first subroutine calculates the value of the objective function, and the second finds the gradient of the objective function. It is only nessecary to recompute the variable terms in the gradient, because the vector is not destroyed by the optimization routine.

The third subroutine calculates $\underline{g}$, the value of the constraints, and the fourth calculates the Jacobian. Again it is only nessecary to recompute the variable elements in the Jacobian.

8.2 Computer experiments.

The program, that is written in FORTRAN IV, has been tested at different problems, and fig. 9 shows some results. The inversion procedure used is an earlier version, that creates slightly more eta-elements than the one described in section 6.2. The idea in section 7.2 has not yet been

	Problem 1	Problem 2
Number of variables	65	136
Number of constraints, linear	44	75
nonlinear	10	41
total	54	116
Number of Jacobi elements, constant	132	263
variable	23	93
total	155	356
Avarage number of elements in the inverse on product form	222	615
Core storage in K bytes:		
User supplied subroutines	4	6
Arrays	13	27
Optimization routine and buffers	69	69
Total	86	102
Execution time in sec CPU:		
Until first feasible solution	0.41	1.70
Pr. iteration after first feasible solution	0.14	0.56
Total time incl input/output, ca.	6	20

Figur 9: Caracteristics of some test problems run on an IBM 370/165 with a FORTRAN H compiler.

implemented.

9. Conclusion.

This paper shows, how it is possible to use the ideas of the GRG-algorithm on large nonlinear models, if special care is taken to reduce the requirement for core storage.

Many methods and ideas from large scale linear programming can be used, directly or in a slightly changed form on the GRG-algorithm, because of the relations between the GRG-algorithm and linear programming.

Although the largest testproblem solved until now has had only 116 constraints, it is reasonable to expect, that problems with 500 constraints and 600 variables can be solved within a region of 250 K bytes.

10. References.

1. Abadie, J. and J. Carpentier: "Generalization of the Wolfe Reduced Gradient Method to the Case of Nonlinear Constraints", in Optimization, R. Fletcher (ed.), Academic Press, 1969.

2. Abadie, J.: "Application of the GRG Algorithm to optimal control problems", in Integer and Nonlinear Programming, J.Abadie (ed.), North-Holland, 1970.

3. Abadie, J.: "Optimization Problems with Coupled Blocks", in Economic Computation and Economic Cypernetics Studies and Research, p 5-26, vol 1970.4.

4. Beale, E.M.L.: "Sparseness in Linear Programming", in Large Sparse Sets of Linear Equations, J.K. Reid (ed.), Academic Press, 1971.

5. Colville, A.R.: "A comparative study of nonlinear programming codes", in Procedings of the Princeton Symposium on Mathematical Programming, H.W. Kuhn (ed.), Princeton University Press, 1970.

6. Fletcher, R. and C.M. Reeves:"Function Minimization by Conjugate Gradients", in British Computer Journal, vol 7, 1964.

7. Ford, L.R. and D.R. Fulkerson: "Flows in Networks", Princeton University Press, 1962.

8. Hellerman, E. and D. Rarick: "Reinversion with the Preassigned Pivot Procedure", in Mathematical Programming, vol 1, 1971.

9. Hellerman, E. and D. Rarick: "The partitioned Preassigned Pivot Procedure", in Sparse Matrices and their Applications, D.J. Rose and R.A. Willoughby (eds.), Plenum Press, 1972.

10. Markowitz, H.M.: "The elimination form of the inverse and its application to Linear Programming", in Management Science, vol 3, 1957.

11. Orchard-Hays, W.: "Advanced Linear Programming Computing Techniques", McGraw-Hill, 1968.

12. Steward, D.V.:"On an approach to techniques for the analysis of the structure of large systems of equations", in SIAM Review, vol 4, 1962.