CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision

(Université de Paris-Dauphine)

Unité Associée au C.N.R.S. n° 825

A PROJECTED GRADIENT METHOD FOR LINEAR PROGRAMMING

CAHIER N° 79
juin 1987

E. JACQUET-LAGREZE

CONTENTS

UNE METHODE DE GRADIENT PROJETE EN PROGRAMMATION LINEAIRE

RESUME

On présente dans ce texte une méthode de gradient projeté conçue spécialement pour résoudre des programmes linéaires.

La **projection n'est pas orthogonale** et est effectuée en utilisant un pivotage sur une matrice de n colonnes et dont le nombre de lignes croît progressivement de 1 à un nombre inférieur ou égal à n (n étant le nombre de variables de décision).

La méthode permet d'obtenir la **solution duale** et permet également d'effectuer une **analyse post-optimale**.

La méthode a été programmée en PASCAL (Turbo Pascal) sur IBM-PC. Elle donne des temps de calcul plus rapides que la méthode révisée du Simplexe.

A PROJECTED GRADIENT METHOD FOR LINEAR PROGAMMING

ABSTRACT

We present in this paper a Projection Gradient Method specially designed to solve linear Programs.

The **projection is not orthogonal** and is done using pivoting rules on a matrix of n columns and for which the number of rows progressively increase from 1 to a number not greater than n (n being the number of initial variables).

The method gives also the **dual solution** and enables to achieve **post-optimality analysis**.

The method has been programmed in PASCAL (Turbo Pascal) on an IBM-PC . The method yields run time apparently significantly faster than the Revised Simplex Method.

1. INTRODUCTION

The idea to use the gradient vector and its projection in order to solve linear programs is both natural and not original. In 1823, FOURIER studied a regression model for which he proposed to minimize the maximum deviation yielding so a linear programming formulation. FOURIER (1826) describes in the following terms his method.

> **Pour atteindre promptement le point inférieur du vase, on élève en un point quelconque du plan horizontal, par exemple à l'origine des x et y, une ordonnée verticale jusqu'à la rencontre du plan le plus élevé, c'est-à-dire que, parmi tous les points d'intersection que l'on trouve sur cette verticale, on choisit le plus distant du plan des x et y. Soit m_1 ce point d'intersection placé sur le plan extrême. On descend sur ce même plan depuis le point m_1 jusqu'à un point m_2 d'une arête du polyèdre et, en suivant cette arête, on descend depuis le point m_2 jusqu'au sommet m_3 commun à trois plans extrêmes. A partir du point m_3 , on continue de descendre suivant une seconde arête jusqu'à un sommet m_4 et l'on continue l'application du même procédé en suivant toujours celle des deux arêtes qui conduit à un sommet moins élevé. On arrive ainsi très prochainement au point le plus bas du polyèdre.**

This text of FOURIER is quoted by A. SCHRIJVER (1985) as the first ideas yielding the famous simplex method clearly stated by DANTZIG (1951).

But if we read carefully the text of FOURIER it is the description of the **projected gradient method.**

We start initially in R^n, improving the value of the objective function moving in the direction of the gradient vector. Then we meet a first constraint which becomes active ("le plan"), we move on this plane on a direction which is actually the first projection of the initial gradient vector, moving so in R^{n-1}. Then we meet a second constraint. Then two things may happen. Either it is more interesting to quit the first constraint and move along the projection of the gradient vector on the second constraint remaining so in R^{n-1}, or it is better to keep both constraints active, moving so on a direction defined in R^{n-2}, an edge ("arête") in the case of three dimensions described by FOURIER. At each iteration of this process we keep at the same level or we reduce the dimension of the space. After a number of iterations we move on edges (R^1) from a vertex to an adjacent vertex as in the simplex algorithm.

Let us remark here that the simplex method is also a projected gradient method. But we impose in that method that at each step the gradient has to be projected on an edge (R^1) instead of a subspace of dimension k, k decreasing from n to 0 if n is the initial dimension of the space.

We consider in this paper an implementation of a projected gradient method as roughly described just above. Although some previous comparisons of the simplex method and the classical projection method developed by ROSEN (1960), made for instance by FAURE and HUARD (1965) tend to show that the simplex method is more efficient, the method presented here and specially developed for Linear Programs seems to behave significantly better both in term of number or iterations and running time than the simplex algorithm even in its advance revised form (more than twice or even three times faster).

Two reasons can explain the difference of results. The first is that our method is designed only for linear programs. Therefore we do not need to implement the search of an optimal move whenever a new direction is computed, and we know that we have to move as far as we can in the direction till we meet (activate) a new constraint. The second which may appear more important is that **we do not implement an orthogonal projection** as suggested in the classical projection gradient method of Rosen, but instead we make a projection such that the new direction is an increasing one, not necesseraly optimal in the sens of minimizing the product scalar of the direction and the gradient of the objective function.

The next section is devoted to the introduction of the mathematical notations and to the projection method. The following section presents the algorithm itself and results on the duality. Section 4 presents some considerations of the computer implementation and the efficiency of the method.

2. FEASIBLE DIRECTIONS AND PROJECTION METHODS

2.1 Preliminary Definitions and Notations

We consider a linear program in this canonical form:

$$
\begin{aligned}
&\text{Max } \mathbf{cx} \\
(1)\quad &\mathbf{Ax} \leq \mathbf{b} \\
&\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}
\end{aligned}
$$

where **l** and **u** are respectively the lower and upper bounds of **x**. We do not impose $\mathbf{x} \geq \mathbf{0}$ and some of the variables may be negative.

Le us denote n (j=1,n) the number of variables, m (i=1,m) the number of constraints, **A** being so a m x n matrix.

Direction

Assume that at the iteration k we get a feasible solution $\mathbf{x}^k$, then it satisfies the constraints of (1). Let us call $\mathbf{d}^k$ a direction of move. The next solution $\mathbf{x}^{k+1}$ is given by :

$$(2) \quad \mathbf{x}^{k+1} = \mathbf{x}^k + t\,\mathbf{d}^k$$

where $t \geq 0$ is a parameter.

For t small enough $\mathbf{x}^{k+1}$ remains feasible if and only if the direction $\mathbf{d}^k$ is feasible .

Feasible direction

Let us consider $\mathbf{A}^\circ$ the submatrix of **A** corresponding to the active constraints at point $\mathbf{x}^k$, $\mathbf{d}^k$ is feasible if it satisfies:

$$(3) \begin{cases} \mathbf{A}^\circ \mathbf{d}^k \leq \mathbf{0} \\ d^k_j \geq 0 \quad \text{if } x^k_j = l_j \\ d^k_j \leq 0 \quad \text{if } x^k_j = u_j \end{cases}$$

When necessary we shall also use slack variables e_i ($\mathbf{A}^\circ\mathbf{d} + \mathbf{e} = \mathbf{0}$).

2.2 The orthogonal projection method of Rosen

Let us consider the gradient vector **c**. At each iteration we want to find a new feasible direction $\mathbf{d}^k$ which is as close as possible to the gradient vector **c** , i.e. whose angle with **c** is minimum. If $\mathbf{d}^k$ is normalized to 1, we get such a direction in solving at each iteration the following program:

$$\begin{aligned} &\text{Max} < \mathbf{d}^k \mathbf{c} \\ &\text{s.t.} \qquad (3) \\ &\| \mathbf{d}^k \| = 1 \end{aligned}$$

It can be shown (see for instance Minoux (1983)), that a solution of this program is given by:

$$(4) \quad \mathbf{d} = (\mathbf{I} - \mathbf{A}^{\circ T} (\mathbf{A}^\circ \mathbf{A}^{\circ T})^{-1} \mathbf{A}^\circ) \mathbf{c}$$

P is then normalized yielding

$$\mathbf{d}^k = \mathbf{d} / \| \mathbf{d} \|$$

Since usually $\mathbf{A}^\circ$ differs at each iteration from only one row, it is possible to update the projection matrix without computing (4) all over again at each iteration.

Nevertheless the computations seem to be costly in running time, compared to the simplicty of some pivoting operation.

The purpose of the non orthogonal projection method developed in the next paragraph is to allow an update of the direction using only simple pivoting rules.

2.3 A non orthogonal projection method

Let us define a matrix **P** with n+1columns and n+1 rows. For computation efficiency it is possible and better to use a matrix with a number of rows which will progressively increase from 1 to a number not greater than n+1, but for the presentation of the method let us use a matrix with n+1 rows.

2.3.1. Definition of the projection matrix P

Let us introduce slack variables z_j for the direction **d** representing the deviation of **d** from the initial gradient vector **c** (z_j is of any sign)

$$(5) \quad \mathbf{d} = \mathbf{c} + \mathbf{z}$$

The first row of **P** contains the scalar product expressed with the variables z_j. Note that we use a similar presentation in the tableau form

of the simplex when we express the objective function in terms or the variables out of the basis.

$$(6) \quad \langle \mathbf{d},\mathbf{c} \rangle = \sum_j d_j c_j = \sum_j c_j^2 + \sum_j c_j z_j$$

Let us put the opposite of the constant term in the column 0 , although it is not necessary to update this value :

$$P_{0,0} = - \sum c_j^2$$

Initially the first row will contain the c_j , each column of P corresponding to a variable z_j : $P_{0j} = c_j$

If we write equation (5) putting the constant terms on the right-hand side we get :

$$\mathbf{d} - \mathbf{z} = \mathbf{c}$$

This equation written in matrix form gives the initial **P** matrix where we put the right-hand side **c** in column 0 , and where we do not write explicitly n columns for vector **d**.

P

i \ j	0	z_1	z_2	. . . .	z_n
0	$\langle d,c \rangle$	c_1	c_2	. . . .	c_n
1	c_1	-1	0	. . . .	0
2	c_2	0	-1	. . . .	0
.	.	.	.	. . . .	.
.	.	.	.	. . . .	.
n	c_n	0	0	. . . .	-1

At first the z_j are present in the matrix and play the role of variables out of a "basis". If we put the value of z_j to 0 we get the corresponding values of **d** in the column 0. Later some of the z_j will be replaced by the slack variables e_j of the active constraints. At any iteration the variables in the columns of **P** will be put to 0 (variables z_j or e_j) in order to get the values of direction **d** .

2.3.2 Preparation of P with a new active constraint

The matrix **P** is modified by a pivoting rule whenever a new constraint i_a becomes active. (Generally the active constraint will be the first one encountered when we move using the previous direction **d**).

The constraint i_a becomes active , so the corresponding slack variable will take place of one of the column variables j_p of the matrix **P** , the corresponding variable (z_j or e_j) playing then the role of an "entering variable into the basis" and having then a positive value that we do not need to take care of.

case of i_a is a constraint $i \leq m$

From (3) and using the corresponding slack variable , we get the equality:

$$(7) \quad \sum_j a_{iaj} d_j + e_{i\,a} = 0$$

(7) has to be expressed in function of the variables of **P** so we need to use classical pivoting rules , once for each row of **P** corresponding to a coefficient a_{iaj} different from 0 since each of these rows gives the value of d_j expressed in terms of the variables of **P**.

case of i_a is a constraint $d_j \geq 0$ (or $d_j \leq 0$)

From (3) and using the corresponding slack variable, we get the equality:

$$(8) \quad d_j - e_{j\,a} = 0 \quad \text{or} \quad (9) \quad d_j + e_{ja} = 0$$

For equation (8) or (9), we do not need any pivoting operation since we can find in **P** the corresponding row showing how d_j is expressed in terms of the variables of **P** .

Whatever the case is, we use equation (7), (8) ,or (9) as an additional working row and we also need a temporary additional column for the slack variable e_{ia} or e_{ja} . This column has 0 everywhere except in the additional row where we put $P_{n+1,n+1} = +1$ or -1 depending of the sign of the coefficient of the slack variable in the additional row (7) , or (8) , or (9) Now the matrix **P** is prepared but before we can perform the pivoting rule

yielding the new direction **d** we need to find which variable of **P** has to quit **P**

2.3.3 Search of the variable which will leave P.

The new direction will differ from the previous one because we have the new constraint active with the corresponding slack variable entering **P** but also because a variable (z_j or e_j) has to leave.

To determine the leaving variable or corresponding pivot column j_p , we try to keep the product scalar <**d**,**c**> as great as possible. In order to avoid long computations, we suggest to use the following criterion (10). Consider rows 0 and n+1 and let us denote w_j the variable associated to column j (w_j = z_j or e_j)

variables (w_j = z_j or e_j)		w_1	w_2	. . .	w_n	e_{ia}
	<**d**,**c**>	P_{01}	P_{02}	. . .	P_{0n}	0
additional row	$P_{n+1,0}$	$P_{n+1,1}$	$P_{n+1,2}$	. . .	$P_{n+1,n}$	$P_{n+1,n+1}$

Find column j among the columns for which the variable is either z_j or e_j, a slack variable of a non binding constraint, for which $P_{0j} * P_{n+1,0} > 0$, such that

$$(10) \quad \max_j \; \text{Sign}(P_{n+1,0}) * P_{0j} / P_{n+1,j}$$

The reason of this criterion is the following. The two rows correspond to equations (11) and (12).

$$(11) \quad \langle \mathbf{d},\mathbf{c} \rangle = \sum_j P_{0j} w_j$$

$$(12) \quad P_{n+1,0} = \sum_j P_{n+1,j} w_j + P_{n+1,n+1} e_{ia}$$

Before the constraint becomes active, e_{ia} is different from 0. The constraint becomes active, e_{ia} will enter in **P** and therefore will have to become null. To achieve this goal, and looking at equation (12) we see that one of the w_j has to become different from 0, and will leave **P** after pivoting.

To maintain equation (12) true, w_j will take the value $(P_{n+1,0} / P_{n+1,j})$ yielding so a variation of $\langle \mathbf{d},\mathbf{c} \rangle$ equal to $P_{0j} * (P_{n+1,0} / P_{n+1,j})$. This value is normaly negative and $\langle \mathbf{d},\mathbf{c} \rangle$ will decrease at each iteration. Since we wish to maximize $\langle \mathbf{d},\mathbf{c} \rangle$, we wish to minimize the amount of the negative decrease, therefore to maximize this negative value. Since $P_{n+1,0}$ does not depend of j, we get criterion (10).

Applying criterion (10) gives a column j_p and the corresponding leaving variable w_{jp} (z_{jp} or e_{jp}).

2.3.4 Pivoting

P is modified using the standard pivoting fomula, the pivot being $P_{n+1,jp}$ After this pivoting operation,the pivot column can be removed from **P,** the additional column can take its place and we move so the column n+1 corresponding to the new variable e_{ia} in place of the pivot column j_p .

We do not use anymore equation (12), therefore we can erase the corresponding row n+1 in **P** in order to leave place for a new active constraint (7) , (8) , or (9) . The additional column n+1 is again free for a new iteration as explained in § 2.3.2.

All the relevants variables are expressed in function of the new set of variables, and in particular the values of **P** are updated in this process giving so the new projected gradient in column 0.

3. THE ALGORITHM

Now that we have described the projection method, we can design different algorithms based on it.

We present in § 3.1 an algorithm which gives an optimal solution whenever we reach a feasible solution. The same algorithm can be used for problems where the initial solution (the origin for instance) is not feasible. We did get an optimal solution for all the problems we solved, starting from the origin, except for a particular example we built in order to show that the optimality condition can sometimes be fullfilled before we reach a feasible solution.

To garantee feasibility in all situations requires nevertheless the adjunction of some artificial variables. This point is not discussed in details in this paper, because we need more experience using the algorithm. Let us just remark that since we work with inequality constraints, we need just one artificial variable for all the inequality constraints.

3.1 Description of the algorithm

Initialization

Choose any feasible or non-feasible solution $\mathbf{x}^0$. It can be the origin, and this choice generally yields better results.

Initialize **P** as done in § 2.3.1. The initial direction is then the gradient vector $\mathbf{d}^0 = \mathbf{c}$.

Find the first active constraint as explained thereafter and the corresponding value of t^0.

Compute $\mathbf{x}^1 = \mathbf{x}^0 + t^0 \mathbf{d}^0$

Iteration k

Update the matrix **P** using the entering active constraint as described in § 2.3.2, 2.3.3 and 2.3.4. We get in row 0 of **P** the projected gradient vector $\mathbf{d}^k$

Determine the new active constraint encountered when moving in direction $\mathbf{d}^k$ and the corresponding minimum value t^k. To do so, we consider all the constraints similar to those of system (3).

Nevertheless, it is not necessary to consider a constraint which is already active, nor a constraint that would necessarily remain satisfied considering a move in direction $\mathbf{d}^k$. We keep the most restrictive of the considered constraints and get the corresponding minimum value t^k.

Compute $\mathbf{x}^{k+1} = \mathbf{x}^k + t^k \mathbf{d}^k$

Optimality test

The test is derived from the Kuhn and Tucker conditions of optimality. It is performed just after the projection of $\mathbf{d}^{k-1}$,that is to say after the updating of **P.** It is rather complex because in some situations we get a null vector for $\mathbf{d}^{k}$ before we get an optimal solution. We need then to choose a new direction. Sometimes we can find an optimal solution in less than n iterations, then some of the z_j remain in **P.**

Let us define $E = \{ j / w_j = e_j \}$ and $Z = \{ j / w_j = z_j \}$, (11) can be expressed by (13) :

$$(13) \qquad \langle \mathbf{d},\mathbf{c} \rangle = \sum_{j \in E} P_{0j}\, e_j + \sum_{j \in Z} P_{0j}\, z_j$$

Suppose first $Z = \emptyset$, then if P_{0j} is negative for all j, we have an optimal solution, unless $w_j = e_j$ and e_j is the slack variable associated to a binding constraint in which case P_{0j} can be of any sign.

If $Z \neq \emptyset$, then if the previous condition remains true for all j belonging to E , and if $x_j = l_j$ for all j belonging to Z, then it is possible to stop the algorithm .

3.2 Duality and post-optimal analysis

The method provides the values of the dual variables.

At the optimum we can interprete equation (13) as giving the dual prices of the constraints.

If $Z = \emptyset$ then the optimal solution is **basic** in the sens of the simplex theory , and we have exactly n active constraints corresponding to n hyperplanes whose intersection give the solution. Then each coefficient P_{0j} gives the symetric of the dual price of the corresponding constraint. A decrease of 1 unit of the slack variable e_j off the frontier, or an increase of 1 unit of ressource b_j which is an equivalent statement, would allow an increase of P_{0j} unit of the economic function.

Therefore if the slack variable e_j corresponds to an active constraint i of system (3) and if we denote y_i the dual variable, then we obtain the dual values using equation (14):

$$(14)\quad y_i = -\,P_{0j} \quad \text{for all j corresponding to the associated constraint i}$$

If $Z \neq \emptyset$ then the optimal solution is **not** a **basic** one which means that the optimal solution is not unique and that we end up with a solution which belongs to a subspace of dimension card(Z), this subspace (a polyedron) being the set of optimal solutions.

P at the optimum enables a post-optimal analysis. In a matrix form we can write :

$$(15)\quad \mathbf{d = P\,w = P_e\,e + P_z\,z}$$

Assume first that $Z = \emptyset$, then let us choose a slack variable e_j corresponding to a non binding constraint and for which we give a positive increase, keeping all the other slack variables to their null level (i.e. keeping all the other constraints active). Doing so we shall go down an edge of the polyedron, leaving the chosen constraint. We shall decrease the objective value according to the dual price P_{0j}. But thanks to (15) we know in which direction we should move in order to perform this post-optimal analysis :

$$(16)\quad d_i = -P_{ij} \quad \text{for } i = 1,n$$

So if $Z = \emptyset$, each of the n columns correspond to the n directions of the edges adjacent to the optimal solution .

If $Z \neq \emptyset$ then either we choose a j belonging to E and we move on a facet yielding a decrease of the objective function, or we choose a j belonging to Z and then we move on a facet of the sub-polyedron of the optimal solutions yielding another optimal solution.

3.3 An illustrative example

Let us illustrate the method on the following example.

$$\text{Max } 2x_1 + x_2 + 8x_3$$

s.t.

$$\begin{aligned} -x_1 - x_2 + x_3 &\le 0.5 \quad c_1 \\ -x_1 - x_2 + 2x_3 &\le 2 \quad c_2 \\ x_1 &\le 2 \quad c_3 \\ x_2 &\le 2 \quad c_4 \\ x_1, x_2, x_3 &\ge 0 \end{aligned}$$

The geometric representation of this LP is given in fig. 1

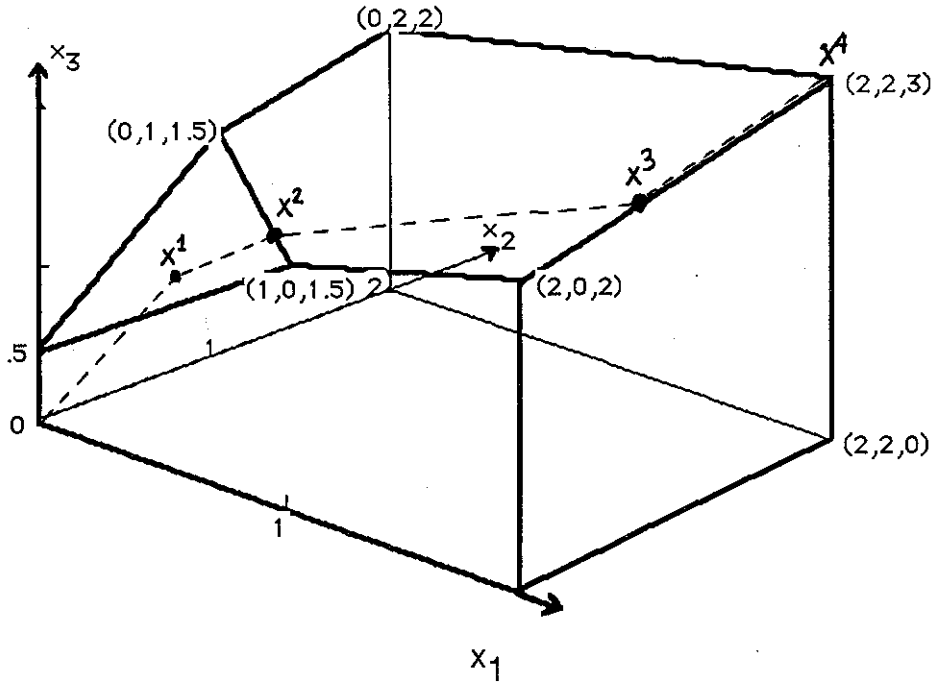

fig 1

The method gives the optimal solution in the 4 following iterations :

Iteration 0

Starting point : $\mathbf{x}^0 = (0,0,0)$

$\mathbf{d}^0 = \mathbf{c} = (2,1,8)$ which gives the following table for **P**

j	0	1	2	3
active contraints				
<**d**,**c**>		2	1	8
i	d			
1	2	-1	0	0
2	1	0	-1	0
3	8	0	0	-1

$\mathbf{d}^0 = (2,1,8)$ which yields solution $\mathbf{x}^1 = (0.2, 0.1, 0.8)$

Iteration 1

After preparing the first active constraint (C_1), **P** becomes

	j	0	1	2	3	4
	variables		z_1	z_2	z_3	e_1
	active contraints					
	<**d**,**c**>		2	1	8	0
	i	d				
	1	2	-1	0	0	0
$\mathbf{P}^0$	2	1	0	-1	0	0
	3	8	0	0	-1	0
	4	5	(1)	1	-1	-1

Applying criterion (10) , we get $j_p = 1$.
After pivoting , we get $\mathbf{P}^1$.

	j	0	1	2	3	4
variables			z_1	z_2	z_3	e_1
active contraints			1			
	<d,c>		0	-1	10	2
	i	d				
	1	7	0	1	-1	-1
$\mathbf{P}^1$	2	1	0	-1	0	0
	3	8	0	0	-1	0
	4	5	1	1	-1	-1

We then remove the pivot column j=1, and copy in its place the additional column 4

	j	0	1	2	3
variables			e_1	z_2	z_3
active contraints			1		
	<d,c>		2	-1	10
	i	d			
	1	7	-1	1	-1
$\mathbf{P}^1$	2	1	0	-1	0
	3	8	0	0	-1

$\mathbf{d}^1$ = (7,1,8) which yields solution $\mathbf{x}^2$ = (0.813, 0.188, 1.5)

Iteration 2

	j	0	1	2	3
variables			e_2	z_2	z_3
active contraints			2		
	<d,c>		2	-1	12
	i	d			
	1	15	-1	1	-2
$\mathbf{P}^2$	2	1	0	-1	0
	3	8	0	0	-1

$\mathbf{d}^2$ = (15,1,8) which yields solution $\mathbf{x}^3$ = (2, 0.267, 2.13)

Iteration 3

	j	0	1	2	3
variables			e_2	e_2	z_3
active contraints			2	3	
	<d,c>		1	-1	10
	i	d			
	1	0	0	1	0
$\mathbf{P}^3$	2	16	-1	-1	-2
	3	8	0	0	-1

$\mathbf{d}^3$ = (0,16,8) which yields solution $\mathbf{x}^4$ = (2, 2,3)

Iteration 4

	j	0	1	2	3
variables			e_2	e_3	e_4
active contraints			2	3	4
	<d,c>		-4	-6	-5
	i	d			
	1	0	0	1	0
$\mathbf{P}^4$	2	0	0	0	1
	3	0	0.5	0.5	0.5

$P_{0j} < 0$ for all j , so the **optimality test** is satified.

The **dual variables** y_i are given in the first row:

$y_1 = 0$ (c_1 is not active) $y_2 = 4$ $y_3 = 6$ $y_4 = 5$

A **post-optimal analysis** can be performed in choosing the directions given in each column of $\mathbf{P}^4$. These directions represent the adjacent edges to the optimal solution. For instance, if we choose direction opposite to (1, 0, 0.5), we quit the constraint C_3 and variable e_3 takes a positive value.

4 Efficiency and Computer implementation

A first version of this algorithm has been programmed on an IBM-PC using TURBO-PASCAL language. The results in term of efficiency seem to be extremely good compared to other computer codes we had. It is not possible to give yet a systematic comparison with the best computer codes existing on a PC and further experimentation is needed. We give some results in § 4.1 and give some information on the program in § 4.2.

4.1 First results

In the following table, **nbv** is the number of bounded variables when these bounds are different from 0 and $+\infty$, and **nel** is the number of elements different from 0 in matrix **A**.

Since it is possible to use the arithmetic co-processor 8087, we give the computer time in seconds for an IBM-PC running at 4.77 MHz and equiped without (8088 only) and with (8087) such a chip. On an AT, computing time are roughly divided by 3.

PROBLEM	n	m	nvb	nel	iterations	time(sec) 8088	time (sec) 8087
J1	3	4	0	8	4	0.2	0.2
LIND2-D	20	22	14	52	21	5.1	2.7
LIND2-P	34	20	0	64	48	13.9	7.4
CUTTING	37	4	0	64	4	1.0	0.7
PLANIF	17	11	0	160	20	10.5	5.0
AIR FRANCE 1	72	65	0	850	22	17.5	12.8
AIR FRANCE 2	295	24	0	3202	29	32.0	
RNUR	64	85	128	363	70	41.2	25.3
KLEE-MINTY	10	10	0	55	19		1.9
KLEE-MINTY	20	20	0	210	39		10.1
PLAN-P	144	97	72	287	142	139 (2'19")	88(1'28")
PLAN-D	169	96	24	311	222	(3'52")	(2'42")

In the name, extention -D (-P) refers to the dual (primal).

PLAN-P has been solved with "TURBO-SIMPLEX", also written in Turbo-Pascal language, in 9'32" and with 271 iterations (103 for phase 1 and 168 for phase 2).

J1 is the problem solved in the previous section.

Some of these problems have {0,1} coefficients in matrix A which enables faster computations and which explains a smaller difference of performance between 8088 and 8087. Problems PLANIF and LIND2 have almost all their coefficients different from 0 and 1, the 8087 version is then roughly twice faster.

We could solve the Klee-Minty problems up to n = 22. We got numerical problems with n greater. The primal forms are solved in **2n-1** iterations instead of $\mathbf{2^n}$-1 iterations with the simplex algorithm. The dual forms are solved in 1 iteration.

With the Hilbert problems we get the same numerical difficulties as with the simplex method.

4.2 Some remarks on the computer program.

In order to experiment the method and also to use it on real problems for AIR-FRANCE and RENAULT, we wrote a TURBO-PASCAL version of the method called **GRADIENT-LP.**

Only the elements of A different from 0 are stored in an array in RAM. This enables to handle large problems and allows more efficiency in speed, especially that we do not modify this array. The upper and lower bounds do not reduce so much computing time.

The matrix **P** is also stored in RAM. To be more efficient, the matrix is built progressively. It has n+1 columns and a number of rows between 1 and n. At each iteration, the number of rows is equal to card(E), (cf. § 3.2). In the illustrative example of § 3.3, the matrix $\mathbf{P}^2$ has still just 1 row at the end of iteration 2.

In order to handle relatively large problems, **P** is defined as a pointer of an array of dimension 300 at most, this enables to handle easily in RAM a matrix with n = 300 . To store P we need 300 * 300 = 90,000 words of 6 bytes , equal to 540 K. With Turbo-Pascal version 8087, a word is stored on 8 bytes so in practice we should not go above n = 250. Remember though that we do not use slack variables, and often just one artificial variable, so the limit refers to the initial decision variables.

Since the method gives the dual variables, we can solve the dual if n » m, in that case we need to store a matrix **P** of dimension m*m only. So practically 250 is the maximum of the smallest value of n and m, the other being easily 3 to 6 times larger. But we have not experimented such large problems yet.

The allocation of the memory for **P** is dynamic since **P** is defined by a pointer, this allows to use this memory space for previous work such as preparing the data of the problem in order to build system (1). We may wish to explicite **A** in order to find redundant constraints for example. If **A** is also defined dynamically with a pointer, then space can be released after the array of the non-null elements has been built.

We plan also to write a FORTRAN version of GRADIENT-LP in order to be able to make more comparisons with the simplex method, since most of the computer codes are written in this langage.

5. CONCLUSIONS

These first results seem very promising.

Research is needed in order to study the mathematical convergence of the projected gradient method as defined in this paper. We believe that the number of iterations should not exceed n+m at least for the true orthogonal projection method of ROSEN, but this is just a conjecture.

More experimentation of the method is needed in order to study the problem of non feasible solutions and how to handle the artificial variable(s). The choice of a starting solution is also related to this problem. Since we do not work only with vertices, it is easier to choose any feasible solution as a starting one. Some experiments revealed how a good starting solution can increase the efficiency or the method.

Since the first versions we wrote in july 1986, the computer code we have now in june 1987 has been considerably improved, both in running times and in precision. We are currently working in order to save memory space. Actually for almost all the real cases we solved, the percentage of non null elements of our "big" matrix **P** is between 5% and 50%. If we manage

in a better way this hollow structure of **P**, we should reduce the space, and may-be the running time.

REFERENCES

Dantzig, G.B. , (1951), "Maximization of a linear function of variables subjected to linear inequalities". T.C. Koopmans (ed.). *Activity Analysis of Production and Allocation,* John Wiley & Sons, New-York, 339-347.

Faure P., Huard P., (1965) "Résolution de programmes mathématiques à fonction non linéaire par la méthode du gradient réduit", *Revue Française de Recherche Opérationnelle,* **36**, 167-206

Fourier J. , (1826), "Analyse des travaux de l'Académie Royale des Sciences pendant l'année 1823, Partie Mathématique. *Histoire de l'Académie Royale des Sciences de l'Institut de France 6, xxix-xli.*

Rosen J., (1960) "The Gradient Projection Method for Nonlinear Programming, I. Linears Constraints", *J. Soc. Indust. Appl. Math.* **8**, 181-197

Schrijver A. (1985), "The New Linear Programming Method of Karmarkar" *CWI NewsLetter, Erasmus University Rotterdam.*