UNIVERSITE DE NANCY I

U.E.R. SCIENCES MATHEMATIQUES

METHODE DU GRADIENT EN PROGRAMMATION LINEAIRE ET QUADRATIQUE

THESE

pour l'obtention du

DOCTORAT DE SPECIALITE D'INFORMATIQUE

SOUTENUE LE 10 DECEMBRE 1977

PAR

Daniel ATLAN

MEMBRES DU JURY

Président : M. J. LEGRAS

Examinateurs : M. P. BOYER

: M. J.P. HATON

UNIVERSITE DE NANCY I

U.E.R. SCIENCES MATHEMATIQUES

METHODE DU GRADIENT EN PROGRAMMATION LINEAIRE ET QUADRATIQUE

THESE

pour l'obtention du

DOCTORAT DE SPECIALITE D'INFORMATIQUE

SOUTENUE LE 10 DECEMBRE 1977

PAR

Daniel ATLAN

MEMBRES DU JURY

Président : M. J. LEGRAS

Examinateurs : M. P. BOYER

: M. J.P. HATON

Je tiens à exprimer ma profonde et respectueuse gratitude à Monsieur J. LEGRAS, Professeur à l'Université de NANCY I, qui a bien voulu me diriger dans ce travail, m'a conseillé et encouragé tout au long de mes recherches.

Je remercie Monsieur P. BOYER, et Monsieur J.P. HATON, qui me font l'honneur de participer au Jury de cette thèse.

En ce qui concerne la partie programmation de cette étude, je dois beaucoup aux chercheurs qui ont constitué la bibliothèque de programmes de l'Université de NANCY I.

Mes remerciements vont également à Madame D. MARCHAND, pour le soin et la gentillesse avec lesquels elle s'est chargée de la réalisation matérielle de cette thèse.

SOMMAIRE

OBJET DE LA THESE

Ce travail a pour objet la résolution par la Méthode du Gradient des problèmes d'optimisation à contraintes linéaires et à fonction objectif linéaire ou quadratique. Les méthodes et algorithmes sont déduits des méthodes générales de résolution des problèmes d'optimisation avec contraintes exposées dans le cours d'Optimisation de Monsieur LEGRAS.

L'Université de NANCY I possède une bibliothèque de programmes d'optimisation permettant de résoudre une classe de problèmes qui englobe les problèmes de programmation linéaire et quadratique. Une partie du travail a consisté à réaliser en langage Fortran des programmes pouvant résoudre particulièrement la catégorie des problèmes visés. A cet effet, nous nous sommes inspirés des techniques et méthodes employées dans les programmes de bibliothèque. Le premier chapitre contient une étude des programmes existants, qui ont été soit modifiés, soit utilisés tels quels.

En ce qui concerne les problèmes de programmation linéaire, ce travail fait suite à la thèse de Monsieur COHEN qui avait établi les algorithmes. Nous avons étendu le champ d'application de ces méthodes à des problèmes à fonction objectif quadratique et nous avons réalisé la programmation.

CHAPITRE I

ETUDE DES MODIFICATIONS APPORTEES PAR LES PARTICULARITES DES PROBLEMES DE PROGRAMMATION LINEAIRE ET QUADRATIQUE AUX PROGRAMMES DE BIBLIOTHEQUE

1°) GENERALITES SUR LA METHODE DU GRADIENT

La méthode du gradient est une méthode itérative déterminant à l'étape $r+1$ un point X_{r+1} tel que $f(X_{r+1}) > f(X_r)$ jusqu'à atteindre l'optimum s'il existe. Les conditions de Kuhn et Tucker (cf. II.2.3) permettent de caractériser l'optimum.

Le point X_{r+1} est déterminé à partir de la valeur du dernier itéré X_r, de la direction de montée Δ_r et d'un accroissement h.

$$X_{r+1} = X_r + \Delta_r \cdot h$$

A chaque itération, la direction Δ_r est une direction de meilleure montée calculée de façon à rester dans le domaine défini par les contraintes.

2°) SIMPLIFICATIONS DUES A LA LINEARITE DES CONTRAINTES :

Les contraintes sont linéaires en X ; on distingue les contraintes bilatérale de la forme $(\varphi_i(X) = 0)$ et les contraintes unilatérales de la forme $(\varphi_j(X) \geq 0)$. Une contrainte unilatérale est saturée lorsque pour un point X donné, l'inégalité devient égalité.

a) Elimination numérique

Soit le système (1) de p équations à n inconnues ($p \leq n$) correspondant aux contraintes bilatérales vérifiées et unilatérales saturées.

$$\Phi (X_B, X_H) = 0 \qquad (1)$$

On est amené à choisir parmi les n variables :

- p variables de base
- n-p variables indépendantes (libres ou hors-base).

On note : X_B le tableau des p variables de base
X_H le tableau des n-p variables hors base.

Le problème est d'exprimer le vecteur X_B en fonction du vecteur X_H.

$$X_B = F (X_H).$$

Lorsque les contraintes ne sont pas linéaires, on utilise un algorithme d'élimination numérique qui à partir des valeurs numériques de X_H permet le calcul de X_B et donc de $F (X_H)$. C'est une technique itérative et la fin des itérations est obtenue lorsque :

$$|\varphi_i (X_B, X_H)| \leq \varepsilon$$

φ_i : contrainte constituant le système (1).
ε : valeur "petite" que l'on se fixe.

Cette technique est mise en œuvre dans le sous-programme ELINUM de bibliothèque.

. Lorsque les contraintes sont linéaires, il suffit de résoudre un système linéaire de p équations à p inconnues. C'est un problème classique qui peut se résoudre par une inversion de matrice.

Le système (1) peut s'écrire sous forme matricielle :

$$P \cdot X_B + R \cdot X_H = E$$

P : matrice carrée $p \times p$.

R : matrice rectangulaire $p \times (n - p)$.

On suppose la matrice P inversible (contraintes non redondantes et bon choix des variables de base). Il est simple d'exprimer X_B en fonction de X_H :

$$X_B = P^{-1} \cdot (E - R \cdot X_H).$$

Ce calcul sera exécuté à l'intérieur du programme correspondant à l'algorithme du gradient réduit "étendu".

Il est donc inutile d'utiliser le sous-programme ELINUM dans le cas des contraintes linéaires. Il y a par conséquent un gain de temps de calcul nécessaire aux itérations successives propres à cette technique d'élimination numérique.

b) Choix du pas h :

Soit le vecteur X_r obtenu à la $r^{ème}$ itération et Δ_r la nouvelle direction de montée, il faut calculer un accroissement h tel que :

$$X_{r+1} = X_r + \Delta_r \cdot h \quad \text{et} \quad F(X_{r+1}) > F(X_r).$$

En bibliothèque, il existe deux sous-programmes ETAP1 et ETAP2 pour le calcul du pas h.

- Le sous-programme ETAP1, à partir d'une valeur h donnée, détermine le point X_{r+1}. Si ce vecteur X_{r+1} ne vérifie pas toutes les contraintes "vérifiées, non saturées", ETAP1 fait appel à un autre sous programme SATUR qui sature une nouvelle contrainte.

Dans le cas contraire (X admissible), le sous-programme teste la relation :

$$f(X_{r+1}) > f(X_r).$$

Si la relation n'est pas vérifiée, il y a diminution du pas. Il y a incident si le pas h devient trop petit.

- Le sous-programme ETAP2 met en œuvre une technique de détermination automatique du pas qui procède en trois phases correspondant au calcul des points Y, Z, U.

Il y a passage d'une phase à l'autre lorsqu'il y a amélioration de la valeur de la fonction objectif. Le vecteur U correspond à un pas h^* déterminé par un polynôme d'interpolation.

ETAP2 fait également appel au sous-programme SATUR dans le cas où une contrainte n'est pas vérifiée.

. La linéarité des contraintes et la spécialité des fonctions objectif nous permet de calculer le pas h de manière exacte sans approximations successives.

Chaque contrainte unilatérale ($\varphi_i(X) > 0$) peut s'écrire sous la forme matricielle :

$$t_{A_i} \cdot X_{r+1} \geq e_i$$

t_{A_i} et X_{r+1} : vecteurs à n composantes.

e_i : scalaire.

En remplaçant X_{r+1} par son expression, on obtient :

$$t_{A_i}(X_r + \Delta_r \cdot h_i) \geq e_i$$

d'où

$$h_i = \frac{e_i - t_{A_i} \cdot X_r}{t_{A_i} \Delta_r} \qquad \text{si} \quad t_{A_i} \Delta_r > 0$$

Afin que le point X_{r+1} reste dans le domaine admissible, h est choisi tel que :

$$h = \underset{i \in K}{\text{Min}}\ h_i$$

K : ensemble des contraintes unilatérales vérifiées non saturée

Ce calcul est mis en œuvre dans le sous-programme SATUR (cf. III. 7)

c) Modification du calcul de la direction de montée dans le cas de la programmation linéaire :

On utilise la méthode du Gradient Réduit "étendu". Cette méthode s'inspire de l'algorithme du Gradient Réduit qui procède pour le calcul de la direction de montée en deux phases :

i) le calcul des coefficients de Lagrange λ :

$$\lambda = [\ {}^t\Phi'_B\ ]^{-1} \cdot [\ f'_B\ ]$$

${}^t\Phi'_B$: matrice carrée ($p \times p$) des contraintes saturées dont les colonnes correspondent aux indices des variables de base.

f'_B : gradient de la fonction objectif en X_B.

u) La direction de montée Δ est calculée à l'aide des formules suivantes :

$$\Delta_H = [\ f'_H\ ] - [\ {}^t\Phi'_H\ ] \cdot [\lambda]$$

$$\Delta_B = 0.$$

Dans la méthode du Gradient Réduit "étendu", on étend le calcul de Δ_H dans l'espace des variables libres (H) à l'espace des variables de base en maintenant saturées les contraintes précédemment saturées. Pour calculer Δ_B on utilise la technique d'élimination vue au paragraphe (2.a).

Δ_B est égal à : $\Delta_B = -[\Phi'_B]^{-1} . [\Phi'_H] . \Delta_H$.

Sous forme matricielle : $\Delta_B = - P^{-1} . R . \Delta_H$.

C'est la linéarité des contraintes qui nous permet cette élimination simple. On a donc modifié le sous-programme GRADUI existant en bibliothèque.

3°) MODIFICATIONS ET PARTICULARITES INTRODUITES PAR LA LINEARITE DE LA FONCTION OBJECTIF :

L'optimum en programmation linéaire possède deux caractéristiques :

- il se trouve sur un des sommets du polyèdre déterminé par les contraintes ;
- il sature exactement n contraintes.

. La première propriété a pour conséquence de calculer le pas h de manière exacte (cf. 2. b) puisque si l'optimum existe, il se trouve à l'un des sommets du polyèdre.

. La seconde caractéristique a influé sur la méthode de résolution. Puisque l'optimum sature exactement n contraintes, le procédé de résolution de problèmes de programmation linéaire passe par une phase dite de "saturation" (cf. II. 7), sature progressivement une contrainte supplémentaire sans en libérer, jusqu'à atteindre le nombre n.

Ayant un point $\overline{X}_0$ réalisable, obtenu lors de la phase "initialisation", si ce point ne sature pas n contraintes, on procède à la phase de "saturation" pour obtenir un nouveau vecteur $\overline{\overline{X}}_i$ qui sature exactement n contraintes.

Ce point $\overline{\overline{X}}_i$ est le vecteur de départ de la phase "optimisation".

4°) MODIFICATIONS ET PARTICULARITES INTRODUITES PAR LA FONCTION OBJECTIF QUADRATIQUE :

a) Calcul du pas h :

L'optimum, s'il existe, en programmation quadratique peut se trouver à l'intérieur ou à la frontière du domaine réalisable. La fonction objectif est du second degré en X. Par conséquent, connaissant un vecteur X_r réalisable et une direction de montée admissible Δ_r, la fonction objectif est une fonction du second degré en h :

$$f(X_r + D_r h) = -\frac{1}{2}\,{}^t(X_r + D_r h)\,.\,C\,.\,(X_r + D_r h) + {}^t p\,.\,(X_r + D_r h$$

Pour calculer h, il suffit de résoudre l'équation du 1er degré en h :

$$\frac{df(X_r + D_r h)}{dh} = 0.$$

Cette équation détermine une valeur h^* de manière exacte. Pour que le point X_{r+1} reste dans le domaine réalisable, on choisit h tel que :

$$h = \text{Min}\,(h^*,\ h_i)$$

h_i : valeur de h déterminé par les contraintes (cf. 2.b).

Ce procédé de calcul est mis en œuvre dans le sous-programme QSATUR (cf. V.5). On n'a pas utilisé les sous-programmes de bibliothèque ETAP1, ETAP2.

b) Calcul de la direction de montée :

On utilise l'algorithme du Gradient Projeté mis en œuvre dans le sous-programme GPROJ de bibliothèque.

Il calcule les coefficients de Kuhn et Tucker et teste leur signe.

Si tous les coefficients sont négatifs ou nuls, le sous-programme calcule la direction de montée :

$$D = f'(X_r) - \sum_i \lambda_i \varphi_i'(X_r)$$

puis teste la norme de D.

. Si $\max |d_K| \leq \varepsilon$ la norme est considérée comme nulle, il y a sortie du sous-programme.

. Si le plus grand des coefficients de Kuhn et Tucker est positif, le sous-programme repère son indice j_1, libère la contrainte correspondante et calcule les coefficients $\bar{\lambda}_i$ puis :

$$\bar{D} = f'(X_r) - \sum_{i \neq j_1} \bar{\lambda}_i \varphi_i'(X_r)$$

Ensuite, il choisit une variable de base acceptable, la libère, puis diminue M (nombre de variables de base) de 1.

Ce sous-programme est utilisé sans modifications majeures.

5°) REMARQUES GENERALES :

a) En ce qui concerne la recherche d'une solution réalisable (phase "initialisation" (III.4)), la méthode de résolution établie est une méthode générale pour tous les problèmes à contraintes linéaires et à fonction objectif quelconque. La recherche d'une solution réalisable ne fait pas intervenir la fonction objectif.

b) Les informations concernant l'état des contraintes (vérifiées ou non, saturées ou non) et des variables (de base ou hors base) sont rangées dans des tableaux ILU et IVL indiquant des valeurs booléennes. (cf. III.1).

c) La gestion des variables de base (cf. III.1) est faite automatiquemen par programme (choix de la variable entrante et sortante). Cette gestion des variables est importante pour l'inversion de la matrice P (matrice des coefficients des contraintes "utiles" correspondant aux indices des variables de base).

d) En programmation quadratique, on a développé une méthode d'accélération de convergence (cf. IV.5) déduite de la méthode des directions conjuguées, ce qui permet de réduire considérablement le nombre d'itérations.

6°) CONVENTIONS :

. On supposera que : - aucune contrainte n'est redondante ;
- toutes les contraintes sont compatibles.

. On se contentera d'étudier les problèmes de maximisation. Si l'on doit minimiser une certaine fonction Z, on se ramènera au cas précédent en cherchant Max (-Z). (Les méthodes de montée développées seront, dans ce dernier cas, des méthodes de descente).

. Par convention, on réserve les p premiers indices aux contraintes bilatérales, ce qui permet de ne pas stocker des informations inutiles.

CHAPITRE II

METHODES ET ALGORITHMES DANS LE CAS D'UNE FONCTION OBJECTIF LINEAIRE

1°) NOTATIONS :

a) Soit $X \in \mathbb{R}^n$; trouver $X^* (x_1^*, x_2^*, \ldots, x_n^*)$ qui rend f (X) maximum.

. f (X) est une fonction linéaire de X.

. X doit vérifier les contraintes linéaires suivantes :

$$\varphi_\ell (X) = 0 \qquad \forall \ell \in L$$

$$\varphi_m (X) \geq 0 \qquad \forall m \in M.$$

L : ensemble des indices des contraintes bilatérales

M : ensemble des indices des contraintes unilatérales.

. Soient p le nombre de contraintes bilatérales et q le nombre de contraintes unilatérales.

b) Notations matricielles :

Soit $X \in \mathbb{R}^n$, trouver X tel que :

$$\text{Max } Z = {}^tC \,.\, X.$$

Avec les contraintes suivantes :

$$\begin{cases} A \,.\, X = E & : \quad p \text{ contraintes bilatérales} \\ A' \,.\, X >> E' & : \quad q \text{ contraintes unilatérales.} \end{cases}$$

${}^tC$: vecteur ligne à n composantes.

A : matrice (p × n) des coefficients des contraintes bilatérales.

A' : matrice (q × n) des coefficients des contraintes unilatérales.

E : vecteur colonne à p composantes.

E' : vecteur colonne à q composantes.

c) Remarques :

. le cas mixte est une généralisation du cas standart (problème avec contraintes bilatérales et contraintes de signe) et du cas canonique (problème avec contraintes unilatérales).

2°) METHODES DU GRADIENT

A) Méthode du Gradient Réduit :

a) Nous n'emploierons cette méthode uniquement dans le cas où le nombre de contraintes saturées est inférieur à n.

Soit $X \in \mathbb{R}^n$, Maximiser $Z = {}^tC \cdot X$ tel que :

$$(P) \quad \begin{cases} A \cdot X = E & : \text{ p contraintes bilatérales} \\ A' \cdot X \geq E' & : \text{ q contraintes unilatérales.} \end{cases}$$

Soit $\overline{X}$ un vecteur réalisable de (P) vérifiant, :

$$\begin{cases} A \cdot \overline{X} = E & (1) \\ A'_J \cdot \overline{X} = E'_J & (2) \\ A'_K \cdot \overline{X} \geq E'_K & (3) \end{cases}$$

J : ensemble des indices des contraintes unilatérales saturées.

K : ensemble des indices des contraintes unilatérales non saturées.

M : $J \cup K$

$p + j$ = nombre de contraintes saturées (bilatérales vérifiées + unilatérales saturées).

On décompose l'ensemble des n indices des variables en deux sous-ensembles :

- B : ensemble des indices des variables de base card (B) = $p + j$.
- H : ensemble des indices des variables libres ou hors base card (H) = $n - (p + j)$.

Le vecteur $X (x_1, x_2, \ldots, x_n)$ peut se décomposer en :

$$X = (X_B, X_H) \quad \text{avec} \quad X_B = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_{p+j}} \end{bmatrix} \qquad \{i_1, i_2, \ldots, i_{p+j}\} = B$$

$$X_H = \begin{bmatrix} x_{i_{p+j+1}} \\ x_{i_{p+j+2}} \\ \vdots \\ x_{i_n} \end{bmatrix} \qquad \{i_{p+j+1}, i_{p+j+2}, \ldots i_n\} = H$$

Les équations (1) et (2) deviennent :

$$\begin{cases} A_B \cdot X_B + A_H \cdot X_H = E & (4) \\ A'_{JB} \cdot X_B + A'_{JH} \cdot X_H = E'_J & (5) \end{cases}$$

On pose :

$$P = \begin{bmatrix} A_B \\ A'_{JB} \end{bmatrix} \qquad R = \begin{bmatrix} A_H \\ A'_{JH} \end{bmatrix} \qquad F = \begin{bmatrix} E \\ E'_J \end{bmatrix}$$

P : matrice carrée $(p + j, p + j)$

R : matrice rectangle $(p + j, n - (p + j))$.

On suppose P inversible, les équations (4) et (5) deviennent :

$$P \cdot X_B + R \cdot X_H = F$$

d'où

$$\boxed{X_B = P^{-1} \cdot (F - R \cdot X_H)} \qquad (6)$$

. L'équation $Z = {}^tC \cdot X$ peut s'écrire :

$$Z = {}^tC_B \cdot X_B + {}^tC_H \cdot X_H$$

D'après (6), l'équation précédente se transforme en :

$$Z = {}^tC_B \cdot P^{-1} \cdot (F - R \cdot X_H) + {}^tC_H \cdot X_H$$

$$Z = {}^tC_B \cdot P^{-1} \cdot F + ({}^tC_H - {}^tC_B \cdot P^{-1} \cdot R) \cdot X_H$$

on pose : $Z_H = {}^tC_B \cdot P^{-1} \cdot R.$

Le Gradient Réduit est égal par définition à : $\left(\frac{\partial Z}{\partial X_h} \right)_{h \in H}$

d'où : $\boxed{\Delta_H = {}^tC_H - Z_H}$

b) Gradient Réduit "étendu" :

Soit $\overline{X'} = \begin{pmatrix} \overline{X'}_B \\ \overline{X'}_H \end{pmatrix}$ le nouveau vecteur réalisable vérifiant les équations (1), (2) et (3).

$$\overline{X'}_H = \overline{X}_H + h \cdot \Delta_H \qquad (h > 0)$$

d'après (6) on doit avoir :

$$\overline{X'}_B = P^{-1} \cdot (F - R \cdot \overline{X'}_H)$$

$$\overline{X'}_B = P^{-1} \cdot F - P^{-1} \cdot R \, (\overline{X}_H + h \cdot \Delta_H)$$

$$\overline{X'}_B = \underbrace{P^{-1} \cdot F - P^{-1} \cdot R \cdot \overline{X}_H}_{\overline{X}_B \text{ (d'après (6))}} - P^{-1} \cdot R \cdot \Delta_H \cdot h$$

$$\overline{X'}_B = \overline{X}_B - P^{-1} \cdot R \cdot \Delta_H \cdot h$$

On pose $\boxed{\Delta_B = -P^{-1} \cdot R \cdot \Delta_H}$

d'où $\overline{X}'_B = \overline{X}_B + h \cdot \Delta_B$.

On appelle "gradient réduit étendu" le vecteur de $\mathbb{R}^n$:

$$\Delta = \begin{bmatrix} \Delta_B \\ \Delta_H \end{bmatrix}$$

$\overline{X}' = \overline{X} + h \cdot \Delta$ vérifie les équations (1) et (2), en effet :

$\forall h \geq 0$: $P \cdot \overline{X}'_B + R \cdot \overline{X}'_H = (P \cdot \overline{X}_B + R \cdot \overline{X}_H) + h \cdot (P \cdot \Delta_B + R \Delta_H)$

$= F + h (-P \cdot P^{-1} \cdot R \cdot \Delta_H + R \Delta_H$

$\overline{X}'$ doit vérifier l'équation (3) :

$A'_K \cdot \overline{X} + h \cdot A'_K \cdot \Delta \geq E'_K$

soit : $h \cdot A'_K \cdot \Delta \geq E'_K - A'_K \cdot \overline{X}$

posons : $-\delta = E'_K - A'_K \cdot \overline{X}$ (7), $-\delta$ est une quantité négative par hypothèse puisque $\overline{X}$ est réalisable.

c) Recherche du pas h :

Le nombre de contraintes saturées étant strictement inférieur à n, on connait une direction de montée Δ, il faut saturer une nouvelle contrainte ; le nouveau point $\overline{X}$ devant rester dans le domaine réalisable. Dans la recherche du pas h, deux cas peuvent se présenter :

- $A'_k \cdot \Delta > 0 \quad , \quad \forall k \in K$

  les équations (3) sont toutes vérifiées pour tout h positif, h peut être choisi aussi grand que l'on veut et le problème n'a pas de solutions finies.

- $\exists k \in K$ tel que : $A'_k \cdot \Delta < 0$

  Posons $K^1 = \{k_1 \in K \mid A'_{k_1} \cdot \Delta < 0\}$

d'après (7) : $\delta_{k_1} = E'_{k_1} - A'_{k_1} \cdot \Delta$

d'où : $h \leq \dfrac{\delta_{k_1}}{A'_{k_1} \cdot \Delta} \qquad \forall k_1 \in K^1$

On a :

$$h = \underset{k_1 \in K^1}{\text{Min}} \left(\frac{\delta_{k_1}}{A'_{k_1} \cdot \Delta} \right)$$

. Le vecteur $\overline{X}' = \overline{X} + h \cdot \Delta$ est réalisable pour (P) et sature une nouvelle contrainte : $(A'_k \cdot X = E'_k \quad (k \in K))$.

d) Algorithme du Gradient Réduit :

L'algorithme peut se décomposer en deux phases :

1ère phase : Calcul de la direction de montée Δ :

Les formules établies au paragraphe précédent nous permettent de calculer Δ :

$$\Delta = \begin{bmatrix} \Delta_B \\ \Delta_H \end{bmatrix}$$

avec

$$\Delta_H = C_H - {}^tC_B \cdot P^{-1} \cdot R$$

$$\Delta_B = - P^{-1} \cdot R \cdot \Delta_H$$

2ème phase : Calcul du pas h.

K : ensemble des indices des contraintes unilatérales saturées

$K^1 = \{k_1 \in K \mid A'_{k_1} \cdot \Delta < 0\}$.

- Si $K^1 = \emptyset$: pas de solutions finies

- Sinon : $h = \underset{k_1 \in K^1}{\text{Min}} \left(\frac{\delta_{k_1}}{A'_{k_1} \cdot \Delta} \right)$

où $\delta_{k_1} = E'_{k_1} - A'_{k_1} \cdot \Delta.$

Soit k_1^* l'indice vérifiant le minimum.

B_r : ensemble des indices des variables de base à l'étape r.

$(\text{Card}\ (B_r) = p + j)$

$\text{Card}\ (B_{r+1}) = p + j + 1.$

B) Méthode du Gradient :

On utilise cette méthode lorsque le nombre de contraintes saturées (bilatérales vérifiées + unilatérales saturées) est égal à n.

a) Caractérisation de l'optimum. Conditions de Kuhn et Tücker :

D'après le théorème de Farkas-Minkowsky, on a :

$$\overrightarrow{\text{grad}}\ f(X^*) = \sum_{\ell \in L} \lambda_\ell \cdot \varphi'_\ell (X^*) + \sum_{j \in J} \mu_j \cdot \varphi'_j (X^*) \qquad (8)$$

L : ensemble des indices des contraintes bilatérales.

J : ensemble des indices des contraintes unilatérales saturées.

λ_ℓ et μ_j sont respectivement appelés les coefficients de Lagrange et, de Kuhn et Tucker.

Conditions de Kuhn et Tücker :

Les conditions nécessaires et suffisantes pour que X^* soit un optimum tous les coefficients μ_j $(\forall\ j \in J)$ doivent être négatifs ou nuls.

L'équation (8) s'écrit sous forme matricielle :

$$C = {}^{t}P \,.\, \Lambda \qquad \text{avec} \quad \Lambda = \begin{bmatrix} \lambda_\ell \\ \mu_j \end{bmatrix} \qquad \begin{matrix} \forall \ell \in L \\ \forall \ell \in J \end{matrix}$$

$$\boxed{\Lambda = {}^{t}P^{-1} \,.\, C} \qquad (9)$$

Remarque : Le calcul de Λ ne s'effectue que dans le cas où le nombre de contraintes saturées est égal à n, par conséquent P est une matrice carrée que l'on peut inverser.

b) Algorithme du Gradient

L'algorithme peut se décomposer en deux phases :

1ère phase : Calcul des coefficients de Kuhn et Tucker ou test :

"X est-il l'optimum ?".

La formule (9) nous donne : $\Lambda = {}^{t}C \,.\, P^{-1}$

P^{-1} : matrice carrée $(n \times n)$.

- Si tous les coefficients μ_j associés aux contraintes unilatérales saturées sont négatifs ou nuls, X^* est l'optimum.

- Si un coefficient μ_j est strictement positif, il faut alors libérer la contrainte unilatérale associée et on passe à la seconde phase.

2ème phase : Le nombre de contraintes saturées est égal à n-1, il faut saturer une nouvelle contrainte, on peut donc appliquer l'algorithme du gradient réduit pour calculer une direction de montée et un pas h ; le nombre variable libre est égal à 1. On se trouve ensuite dans les conditions de la phase 1.

3°) METHODE DE RESOLUTION D'UN PROGRAMME LINEAIRE SOUS FORME MIXTE PAR LA METHODE DU GRADIENT :

Soit (P) un programme linéaire sous forme mixte :

$$(P) \quad \begin{cases} AX = B \\ A'X \gg B' \\ \text{Max } C \,.\, X \end{cases}$$

On distinguera trois phases dans la méthode de résolution :

- Une phase d'initialisation :

A partir d'un vecteur $\vec{\xi}$ quelconque, déterminer un vecteur $\vec{X}$ réalisable.

- Une phase de saturation :

A partir d'un vecteur réalisable $\vec{X}$ qui sature moins de n contraintes déterminer un vecteur $\vec{X}'$, réalisable, qui sature exactement n contraintes

- Une phase d'optimisation :

A partir d'une solution $\vec{X}'$, réalisable, qui sature n contraintes exactement, déterminer la solution optimale X^*, si elle existe.

a) Phase d'initialisation :

Soit ξ^o un vecteur tel que :

$$(P) \quad \begin{cases} A_{\ell_1} \,.\, \xi^o = B_{\ell_1} \\ A_{\ell_2} \,.\, \xi^o > B_{\ell_2} \\ A_{\ell_3} \,.\, \xi^o < B_{\ell_3} \\ A'_{k_1} \,.\, \xi^o \gg B'_{k_1} \\ A'_{k_2} \,.\, \xi^o \ll B'_{k_2} \end{cases} \qquad \begin{array}{l} \text{où : } \ell_1 \cup \ell_2 \cup \ell_3 = L \\ \\ \\ \text{où : } k_1 \cup k_2 = K \\ \\ \end{array}$$

. Si $\ell_2 = \ell_3 = k_2 = \emptyset$, ξ^o est un vecteur réalisable de (P).

. Si l'un des trois sous-ensembles ℓ_2, ℓ_3 et k_2 est non vide, on considère le problème auxiliaire suivant : $(P\ A_0)$.

$$(PA_0) \begin{cases} A_{\ell_1} \cdot X = B_{\ell_1} \\ A_{\ell_2} \cdot X >> B_{\ell_2} \\ -A_{\ell_3} \cdot X >> B_{\ell_3} \\ A'_{k_1} >> B'_{k_1} \\ -A'_{k_2} >> -B'_{k_2} \\ \text{Max} \left(\sum_{k \in k_2} A_k X + \sum_{\ell \in \ell_3} A_\ell \cdot X - \sum_{\ell \in \ell_2} A_\ell \cdot X \right) \end{cases}$$

ξ^o est un vecteur réalisable de $(P\ A_0)$.

Deux cas peuvent se présenter :

<u>1er cas</u> : ξ^o sature moins de n contraintes.

On applique l'algorithme du Gradient Réduit au problème auxiliaire $(P\ A_0)$ jusqu'à saturer n contraintes où jusqu'à trouver une solution réalisable c'est-à-dire que ℓ_2, ℓ_3 et k_2 soient vides.

La fonction objectif de chaque problème auxiliaire sera modifiée à chaque itération suivant les indices des ensembles ℓ_2, ℓ_3 et k_2.

<u>2ème cas</u> : ξ^o sature exactement n contraintes.

. On applique l'algorithme du gradient jusqu'à trouver une solution réalisable, en libérant la contrainte correspondante au coefficient de Kuhn et Tucker positif, et en saturant une nouvelle contrainte.

. Si toutefois les conditions de Kuhn et Tucker sont vérifiées et les ensembles ℓ_2, ℓ_3 et k_2 ne sont pas vides, alors le problème (P) n'a pas de solution.

b) Phase de saturation :

Soit $\vec{X}_0$ un vecteur réalisable obtenu à l'issue de la phase précédente qui sature p + q contraintes (p + q < n).

On applique l'algorithme du Gradient Réduit qui nous donne un nouveau vecteur réalisable $\vec{X}^1$ qui sature p + q + 1 contraintes.

On applique cet algorithme un nombre de fois égal à n - (p + q) jusqu'à obtenir n contraintes saturées et passer à la phase d'optimisation.

c) Phase d'optimisation :

Soit $\vec{X}'_0$ un vecteur obtenu à l'issue de la phase "saturation" qui sature exactement n contraintes.

1er cas : Les conditions de Kuhn et Tucker sont vérifiées : $X^* = \vec{X}'^0$ est solution optimale.

2ème cas : Les conditions de Kuhn et Tucker ne sont pas vérifiées, on applique l'algorithme du Gradient.

4°) EVOLUTION DE LA MATRICE P_r^{-1} :

Le calcul de la direction de montée Δ nécessite à chaque itération la connaissance de la matrice P^{-1}. Pour éviter une inversion de matrice, on propose de faire évoluer celle-ci à chaque fois que l'on libère, ou l'on sature une contrainte. A l'étape r + 1, P_{r+1}^{-1} se calcule à partir de P_r^{-1}.

A l'étape initiale, l'inverse de la matrice P sera calculée par un sous-programme d'inversion de matrice.

Bien entendu, l'utilisateur des programmes pourra inverser la matrice P à chaque itération s'il le désire au lieu de faire évoluer P_r^{-1} comme nous le présentons ci-dessus.

a) Libération d'une contrainte :

On se place dans la 2ème phase de l'algorithme du Gradient où il y a au moins un coefficient de Kuhn et Tücker strictement positif, ce qui implique la libération de la contrainte associée à l'indice du coefficient et la sortie d'une variable de la base B.

Du point de vue de la matrice P, on supprime une ligne et une colonne correspondant respectivement à la contrainte libérée et à la variable sortante.

Procédé du calcul :

Soit une matrice carrée d'ordre n : P (n × n), que l'on partitionne :

$$P = \left[\begin{array}{c|c} P_0 & D \\ \hline {}^tC & e \end{array}\right]$$

P_0 : matrice carrée (n-1, n-1)

${}^tC$: vecteur ligne à n-1 composantes.

D : vecteur colonne à n-1 composantes.

e : scalaire.

On connaît :

$$P^{-1} = \left[\begin{array}{c|c} A & U \\ \hline {}^tL & y \end{array}\right]$$

A : matrice carrée (n-1, n-1)

${}^tL$: vecteur ligne à n-1 composantes.

U : vecteur colonne à n-1 composantes.

y : scalaire.

On cherche l'inverse de la matrice P_0 sous matrice de P.

On calcule :

$$P \cdot P^{-1} = \left[\begin{array}{c|c} P_0 & D \\ \hline {}^tC & e \end{array}\right] \left[\begin{array}{c|c} A & U \\ \hline {}^tL & y \end{array}\right]$$

En développant le calcul on obtient :

$$\left[\begin{array}{c|c} P_0 \cdot A + D \cdot {}^tL & P_0 \cdot U + D\, y \\ \hline {}^tC \cdot A + {}^tL \cdot e & {}^tC \cdot U + e\, y \end{array}\right] = \left[\begin{array}{c|c} I_{n-1} & 0 \\ \hline 0 & 1 \end{array}\right]$$

Ce qui nous amène à résoudre le système d'équations suivant :

$$\begin{cases} P_0 \cdot A + D \cdot {}^tL = I_{n-1} & (1) \\ P_0 \cdot U + D \cdot y = 0 & (2) \\ {}^tC \cdot A + {}^tL \cdot e = 0 \\ {}^tC \cdot U + e \cdot y = 1. \end{cases}$$

De l'équation (2), on déduit le vecteur D :

$$D = - \frac{P_0 \cdot U}{y} \quad \text{si} \quad y \neq 0$$

En remplaçant dans l'équation (1) :

$$P_0 \cdot A - \frac{P_0 \cdot U \cdot {}^tL}{y} = I_{n-1}$$

$$P_0 \cdot \left(A - \frac{U \cdot {}^tL}{y}\right) = I_{n-1}$$

d'où :

$$\boxed{P_0^{-1} = A - \frac{U \cdot {}^tL}{y}} \qquad (3)$$

Remarques : . La matrice $U \cdot {}^tL$ est carrée (n-1, n-1) puisque produit du vecteur U (n-1, 1) par ${}^tL$ (1, n-1).

. La formule (3) est utilisée dans le cas où la contrainte à libérer a pour indice n ainsi que la variable sortante.

. Etudions le cas où la contrainte à libérer a pour indice i ($1 \leq i \leq n$) et la variable sortante a pour indice j ($1 \leq j \leq n$).

On partitionne la matrice P de la façon suivante :

$$P = \left[\begin{array}{c|c|c} P_1 & C_j & P_2 \\ \hline L_i & P_{ij} & \\ \hline P_3 & & P_4 \end{array}\right] \leftarrow \text{ième ligne}$$

jème colonne

On va se ramener au cas précédent en multipliant la matrice à droite et à gauche par respectivement les matrices J_1 et J_2.

$$J_1 = \left[\begin{array}{c|c|c} I_{i-1} & \begin{matrix}0\\0\end{matrix} & O \\ \hline O & \vdots & I_{n-i} \\ \hline 0 \ldots\ldots & 1 & \ldots\ldots 0 \end{array}\right]$$

ième colonne

$$J_2 = \left[\begin{array}{c|c|c} I_{j-1} & O & \begin{matrix}0\\\vdots\end{matrix} \\ \hline 0 \ldots\ldots & \ldots\ldots & 1 \\ \hline O & I_{n-j} & \begin{matrix}\vdots\\0\end{matrix} \end{array}\right]$$

On vérifie facilement : $J_1 \cdot {}^tJ_1 = I_i$ et $J_2 \cdot {}^tJ_2 = I_n$.

J_1 et J_2 sont donc deux matrices orthogonales :

$${}^tJ_1 = J_1^{-1} \quad , \quad {}^tJ_2 = J_2^{-1}$$

On a :

$$J_1 \cdot P \cdot J_2 = \left[\begin{array}{cc|c} P_1 & P_2 & \\ & & C_j \\ P_3 & P_4 & \\ \hline L_i & & P_{ij} \end{array}\right]$$

p_{ij} : scalaire est le coefficient de la matrice P de la ième ligne et jème colonne.

Ce qui nous intéresse c'est l'inverse de :

$$P_0 = \begin{bmatrix} P_1 & P_3 \\ P_3 & P_4 \end{bmatrix}$$

On a : $(J_1 \cdot P \cdot J_2)^{-1} = {}^tJ_2 \cdot P^{-1} \cdot {}^tJ_1$

Si :

$$P^{-1} = \begin{bmatrix} A_1 & C_i & A_2 \\ L_j & & \\ A_3 & & A_4 \end{bmatrix}$$

← jième ligne

↑ ième colonne

$${}^tJ_2 \cdot P^{-1} \cdot {}^tJ_1 = \begin{bmatrix} A_1 & A_2 & \\ A_3 & A_4 & C_i \\ L_j & & p_{ji}^{-1} \end{bmatrix}$$

On pose :

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}$$

$U = C_i$; ${}^tL = L_j$; $y = p_{ji}^{-1}$ dans la formule (3).

Autre formulation :

Soit P_r^{-1} la matrice d'ordre n ou $P^{-1}(k, \ell)$ $\forall k, \ell \in \{1, \ldots n\}$

on cherche P_{r+1}^{-1} où P_{r+1} est déduite de P_r en supprimant la ligne i et la colonne j.

$$\forall k,\ \ell \in \{1,\dots,n\} \quad P_{r+1}^{-1}(k,\ell) = P_r^{-1}(k,\ell) - \frac{P_r^{-1}(j,\ \ell) \times P_r^{-1}(k,\ i)}{P_r^{-1}(j,\ i)}$$

$$\text{si} \quad P^{-1}(j,\ i) \neq 0$$

On supprime la ligne j et la colonne i constituée de zéros et on obtient P_r^{-1}, d'ordre n-1.

Remarque : Ce calcul est utilisé dans le sous-programme GPRØJ (III, 6).

b) Saturation d'une contrainte :

On se place dans la phase 2 de l'algorithme du Gradient Réduit, on a une direction de montée Δ, on calcule le pas h qui nous amène à saturer une nouvelle contrainte.

Procédé de calcul :

La matrice P_r^{-1} est d'ordre k, on cherche l'inverse de P_{r+1} d'ordre k + 1.

La matrice P_{r+1} est déduite de la matrice P_r en ajoutant une ligne composée des coefficients de la contrainte saturée et une colonne correspondant à la variable entrant dans la base. Nous décomposons P_{r+1} et P_{r+1}^{-1} en sous-matrice suivant le même procédé que dans le paragraphe précédent.

Soit :

$$P_{r+1} = \begin{bmatrix} P_r & U \\ {}^tV & y \end{bmatrix}$$

U : vecteur colonne à k composantes.

${}^tV$: vecteur ligne à k composantes.

y : scalaire

Posons :

$$P_{r+1}^{-1} = \left[\begin{array}{c|c} A & C \\ \hline {}^{t}L & e \end{array}\right]$$

Explicitons le calcul de $P_{r+1} \cdot P_{r+1}^{-1} = I_{k+1}$

$$P_{r+1} \cdot P_{r+1}^{-1} = \left[\begin{array}{c|c} P_r \cdot A + U \cdot {}^{t}L & P_r \cdot C + U \cdot e \\ \hline {}^{t}V \cdot A + y \cdot {}^{t}L & {}^{t}V \cdot C + y\, e \end{array}\right] = \left[\begin{array}{c|c} I_{k-1} & O \\ \hline O & 1 \end{array}\right]$$

On obtient le système suivant :

$$\begin{cases} P_r \cdot A + U \cdot {}^{t}L = I_{k-1} & (4) \\ P_r \cdot C + U \cdot e = 0 & (5) \\ {}^{t}V \cdot A + y \cdot {}^{t}L = 0 & (6) \\ {}^{t}V \cdot C + y \cdot e = 1 & (7) \end{cases}$$

De l'équation (5), on déduit : $\underline{C = - P_r^{-1} \cdot U \cdot e}$

De l'équation (7), on déduit : $\underline{e = \dfrac{1}{y - {}^{t}V \cdot P_r^{-1} U}}$ si $(y - {}^{t}V \cdot P_r^{-1} \cdot U \neq 0)$

De l'équation (4) on tire ${}^{t}L$:

$${}^{t}L = - \frac{{}^{t}V \cdot A}{y} \quad (\text{si } y \neq 0). \qquad (8)$$

En multipliant l'équation (4) par P_r^{-1} à gauche, on obtient :

$$A = P_r^{-1} - P_r^{-1} \cdot U \cdot {}^{t}L \qquad (9)$$

(8) et (9) nous donnent : $${}^{t}L = \frac{-\,{}^{t}V \,.\, P_r^{-1}}{y - {}^{t}V \,.\, P_r^{-1} \,.\, U}$$

ou en fonction de e : $$\underline{{}^{t}L = -\,{}^{t}V \,.\, P_r^{-1} \,.\, e}$$

Les équations (4) et (5) nous donnent l'expression de A.

$$\underline{A = P_r^{-1} + \frac{C \,.\, {}^{t}L}{e}} \qquad \text{si } (e \neq 0)$$

Donc :

$$P_{r+1}^{-1} = \left[\begin{array}{c|c} P_r^{-1} + \dfrac{C\,{}^{t}L}{e} & C \\ \hline {}^{t}L & e \end{array}\right]$$

On calculera les valeurs de e, C, ${}^{t}L$ et A dans cet ordre :

$$e = \frac{1}{y - {}^{t}V \,.\, P_r^{-1} \,.\, U} \qquad \text{si } y - {}^{t}V\, P_r^{-1} \,.\, U \neq 0$$

$$C = -\,P_r^{-1} \cdot U \,.\, e$$

$${}^{t}L = -\,{}^{t}V\; P_r^{-1} \,.\, e$$

$$A = P_r^{-1} + \frac{C \,.\, {}^{t}L}{e}$$

Remarque : . Le calcul de l'inverse de la matrice P_{r+1} est possible si la quantité $y - {}^{t}V \,.\, P_r^{-1} \,.\, U$ est non nulle auquel cas le choix de la variable entrante est bon, sinon il faut choisir une autre variable.

. Le calcul précédent a été fait dans le cas où l'indice de la ligne et de la colonne entrante sont égaux à r+1.

Si l'indice de la ligne est égal à i ($1 \leq i \leq r+1$) et l'indice de la colonne à j ($1 \leq j \leq r+1$), on procède de la même manière que dans le cas de la libération d'une contrainte.

Ce qui revient à multiplier par les matrices J_1 et J_2 la matrice P_{r+1} pour se ramener au cas précédent.

$$(J_2 \cdot P_{r+1} \cdot J_1)^{-1} = {}^tJ_1 \cdot P_{r+1}^{-1} \cdot {}^tJ_2$$

. Pratiquement, il faut permuter les lignes r+1 et j avec les colonnes r + 1 et i.

. Ces formules sont mises en œuvre dans le sous-programme SATUR (cf. III, 7).

CHAPITRE III

NOTICE D'UTILISATION DU PROGRAMME GRPL (METHODE DU GRADIENT EN PROGRAMMATION LINEAIRE)

1°) GESTION DES VARIABLES ET DE L'ETAT DES CONTRAINTES :

a) Pour repérer la nature des variables (de base ou hors base), ainsi que l'état des contraintes bilatérales et unilatérales au cours de l'exécution du programme, nous utilisons deux tableaux :

. le tableau d'identificateur IVL (indice des variables libres) où :

$$\begin{cases} IVL\ (B) = 1 & \text{si } x_b \text{ est variable de base.} \\ IVL\ (H) = 0 & \text{si } x_h \text{ est variable libre.} \end{cases}$$

. le tableau d'identificateur ILU (indice des liaisons "utiles") où :

$$\begin{cases} ILU\ (I) = 1 & \text{. si la contrainte } \varphi_i = 0 \text{ est contrainte bilatérale vérifiée ou unilatérale saturée.} \\ ILU\ (I) = 0 & \text{. si la contrainte } \varphi_i > 0 \text{ est une contrainte bilatérale non vérifiée ;} \\ & \text{. si la contrainte } \varphi_i > 0 \text{ est une contrainte unilatérale vérifiée non saturée} \\ ILU\ (I) = -1 & \text{. si la contrainte } \varphi_i < 0 \text{ est une contrainte bilatérale ou unilatérale non vérifiée} \end{cases}$$

Ces deux tableaux seront initialisés par le sous-programme INIT et leur évolution sera assurée par les autres sous-programmes.

b) Changements de base - Tableau IVX :

Il y a changement de base lorsqu'il y a modification de l'état des contraintes :

1er cas : saturation d'une nouvelle contrainte, ce qui entraîne l'augmentation du nombre des variables de base ; une variable hors base entre dans la base.

2ème cas : libération d'une contrainte, ce qui entraîne une diminution des variables de base : l'une d'elle devient variable hors base.

Une des difficultés du choix d'une variable entrante ou sortante vient de la nature de contrainte de type suivant :

$$\varphi_i = a\, x_j + b \geq 0$$

Contrainte $\varphi_i \geq 0$ ne dépendant que de la variable x_j.

. Si une contrainte φ_i est saturée (programme SATUR) et si cette contrainte est de la forme $\varphi_i = a\, x_j + b$, la variable x_j a nécessairement la valeur $-b/a$ et devra devenir variable de base si elle était précédemment hors base.

. Si une contrainte φ_{i_1} est libérée (programme GPROJ) autre que φ_i, la variable x_j ne pourra pas être choisie comme variable libre : une variable x_j, de base ne pourra devenir variable hors base que s'il n'existe aucune contrainte saturée fonction de la seule variable x_j.

Pour que les programmes GPROJ et SATUR possèdent les informations nécessaires aux traitements que nous venons de décrire, nous introduisons le tableau IVX où :

$\begin{cases} IVX(I) = J & \text{si la contrainte } \varphi_i \text{ dépend de la seule variable } x_j \\ IVX(I) = 0 & \text{dans les autres cas.} \end{cases}$

Ce tableau sera initialisé par le programme principal et reste invariant au cours du traitement.

Dans le second cas, une variable x_j ne pourra sortir de la base que si la relation :

$$ILU(I) = 1 \quad \text{et} \quad IVX(J) = J$$

a la valeur "fausse" quelque soit I $(1 \leq I \leq n)$.

c) Des cas particuliers dus aux coefficients des contraintes ne permettent pas un choix arbitraire des variables entrantes ou sortantes de la base.

Par exemple si les deux contraintes suivantes sont saturées :

$$x_2 + x_3 \geq -2$$

$$x_2 + 2x_3 - x_4 \geq 4.$$

Si l'on choisit x_1 comme variable de base, la matrice P aura une colonne de zéros et ne sera pas inversible. Il faut donc tester si le coefficient correspondant à la contrainte saturée de la variable choisie est différent de zéro.

A l'initialisation, le choix est fait automatiquement par programme de manière à ce que la matrice P ne comporte ni lignes, ni colonnes de zéros.

Dans l'évolution de la matrice P^{-1} (cf. III.4), dans le cas d'une saturation de contrainte (respectivement libération de contrainte), c'est-à-dire le passage du rang d'ordre m au rang supérieur m+1 (respect. au rang inférieur m-1), le test d'un coefficient ou "pivot" devant être différent de zéro, permet de détecter si le choix de la variable entrante (resp. sortante) dans la base est mauvais; dans ce cas il est corrigé.

2°) LISTE DES TABLEAUX ET VARIABLES UTILISES

a) Tous les variables et tableaux sont transmis entre les différents sous-programmes par des instructions "COMMON".

Variables simples :

N	nombre de variables.
NT	nombre total de contraintes.
P	nombre de contraintes bilatérales, doit être déclaré "entier".
M	nombre de contraintes "utiles" (bilatérales vérifiées et unilatérales saturées) ; est également le nombre de variables de base.
NA	paramètre de surdimensionnement, peut être choisi arbitrairement égal ou supérieur à N et NT. Indispensable pour MCADIR.
INCI	Indicateur d'incident. Il a la valeur 0 après exécution d'un sous-programme en cas de déroulement normal. En cas d'incident, il prend une valeur entière qui dépend du sous-programme et de la nature de l'incident.
IFI	Indicateur de fin d'itération. Il intervient dans GPROJ et prend la valeur 0 lorsque le maximum est atteint. Sinon il a la valeur 1.
INI	Indicateur égal à 1 si le programme se trouve dans la phase "initialisation" c'est-à-dire qu'une solution réalisable n'a pas encore été trouvée, il a la valeur 0 sinon. Il intervient dans le sous-programme INIT.
H	Pas de montée, évolue dans le programme (sous-programme SATUR).
FX	Contient en permanence la valeur de dernier itéré.
ITER	Compteur d'itérations.

Tableaux :

Nom	Dimension	
X	N ou NA	Contient en permanence le dernier itéré X^r d'éléments $x_1^r, x_2^r, \ldots, x_n^r$. Peut être initialisé par l'utilisateur ou par le programme principal.
D	N ou NA	Contient la direction de montée.
ILU	NT ou NA	Caractérise l'état des liaisons ou contraintes, ILU (I) = { 0 . si bilatérale non vérifiée positive ou unilatérale vérifiée 1 . si bilatérale vérifiée ou unilatérale saturée -1 . si bilatérale non vérifiée négative ou unilatérale non vérifiée négative.
IVL	N ou NA	Caractérise la nature des variables IVL (H) = 0 si x_h est variable hors base. IVL (B) = 1 si x_b est variable de base.
IVX	NT ou NA	Caractérise la nature de certaines contraintes. IVX (I) = J si φ_i dépend de la seule variable x_j. IVX (I) = 0 dans les autres cas.
EPS	2	ε_1 et ε_2 utilisées dans les tests des sous-programmes GPROJ et INIT.
C1	N ou NA	Tableau contenant la dérivée de la fonction objectif. $C = \frac{\partial f}{\partial x_r}$.
G	N × N	Contient les dérivées $\partial \varphi_i / \partial x_s$ des contraintes "utiles" (i ∈ L ou J) par rapport à toutes les variables.
W	N × N	Contient l'inverse de la matrice P des coefficients des contraintes utiles correspondant aux variables de base.

Nom	Dimension	
		Contient les dérivées $\partial \varphi_i / \partial x_s$ de toutes les contraintes par rapport à toutes les variables. Ce tableau est inutile si cette information se trouve sur mémoire périphérique. Pour l'implantation sur petite machine ou pour des problèmes à grand nombre de contraintes, un support externe de mémorisation est indispensable.
IP, T1, T2, C	NT ou NA	Tableaux auxiliaires.

b) Données et paramètres d'entrée :

La lecture des données suivantes est faite au début du programme principal.

- NA, N, NT et P.
- C1 : les coefficients de la fonction objectif.
- IVX : information sur la nature des contraintes.
- V (NT, N) : coefficients des contraintes ; dans le cas où l'on ne veut pas garder en mémoire centrale ce tableau, on lira ces informations sur mémoire externe à chaque fois qu'il sera nécessaire de les utiliser.
- X : vecteur du départ est initialisé par l'utilisateur.

3°) PROGRAMME PRINCIPAL

a) Instructions de déclaration :

```
COMMON/A/ P, N, NT, NA, M, INCI, IFI, P1, N1, D (N), ILU (NT),
          IVL (N), IVX (NT), EPS (2), C (N), IP (N), G (N,N),
          W (N,N), T1 (N), T2 (N), C1 (N)

COMMON /A1/ X(N), FX, H, V (NT,N), ITER, INI
INTEGER     P, P1.
```

b) Composition du programme :

Le programme comprend un programme principal et six sous-programmes.

Deux sous-programmes ARRAY et MINV existant en bibliothèque, permettent d'inverser initialement la matrice W (appelée P dans le chapitre II) ;

Quatre sous-programmes :

- GRADUI : calcul de la direction de montée Δ.
- GPROJ : représente l'algorithme du gradient
- INIT : traduit la phase "initialisation" ou la recherche d'une solution réalisable.
- SATUR : calcul d'un nouvel itéré X_{r+1}.

L'utilisateur se charge des éditions des résultats. Certaines éditions indispensables sont indiquées.

c) Programme principal :

- Instructions de déclaration :
  COMMON /A/ ...
  COMMON /A1/ ...
- Lecture des données
- Phase "initialisation" :

```
      N1 = N + 1
      P1 = P + 1
 5    CALL INIT
      IF (INCI.NE.0) GØ TØ 30
      IF (M.LE.0) GØ TØ 10          <-- test si aucune contrainte saturée
                                        il n'y a pas d'inversion de ma-
                                        trice
      CALL ARRAY (2,M,M,NA,NA,W,W)  }
      CALL MINV (W,M,D1,T1,T2)      } Inversion de la matrice W.
      CALL ARRAY (1,M,M,NA,NA,W,W)  }
      IF (ABS (D1).GT.EPS1) GØ TØ 10    test sur le déterminant
      INCI = 10
      GØ TØ 30
10    IF (INI.NE.0) GØ TØ 20
      DØ 15  I = 1,N                Si la phase initialisation est ter-
15    C (I) = C1 (I)                minée (INI = 0) on traite le pro-
                                    blème initial avec sa fonction
                                    objectif. (C1 (N))
20    CALL GPROJ
      IF (INCI.NE.0) GØ TØ 30
      IF (IFI.LE.0.AND.INI.EQ.1) GØ TØ 40
      CALL SATUR
      IF (INI.EQ.1) GØ TØ 5
      GØ TØ 20
30    WRITE (6,630) INCI
630   FØRMAT (5X, 11HINCIDENT NO,I4)
      STØP 1
40    WRITE (6,640)
640   FØRMAT (5X, 26H1 PAS DE SOLUTION REALISABLE)
      STØP 2
      END
```

. INCI = 10 : le déterminant de la matrice W est nul.

. Sous-programmes appelés : ARRAY, MINV, GPROJ, SATUR.

4°) PHASE "INITIALISATION" - SOUS-PROGRAMME INIT :

Ce sous-programme traduit la phase "initialisation" ou la recherche d'une solution réalisable. (II.1.a).

. Il initialise le tableau ILU sur l'état des contraintes.

$$ILU(I) = \begin{cases} 0 & \text{. si contrainte unilatérale vérifiée} \\ 1 & \text{. si contrainte bilatérale non vérifiée positive} \\ 1 & \text{. si contrainte saturée (bilatérale ou unilatérale)} \\ -1 & \text{. si contrainte non vérifiée négative (bilatérale ou unilatérale).} \end{cases}$$

. Si une contrainte au moins n'est pas vérifiée, l'indicateur INI prend la valeur 1 pour signaler que le vecteur X n'est pas solution réalisable. Dans ce cas, il met en place le problème auxiliaire en calculant les coefficients de la fonction objectif du problème auxiliaire. Dans le cas contraire, INI prend la valeur 0.

. Le tableau IVL indiquant la nature des variables (base ou hors base) est initialisé de manière à ce que la matrice P soit inversible.

$$IVL(^{T}) = \begin{cases} 0 & \text{variable } x_j \text{ hors base.} \\ 1 & \text{variable } x_j \text{ de base.} \end{cases}$$

Paramètres d'entrée :

X : est initialisé par le programme principal.

SUBROUTINE INIT

Instructions de déclarations :

CØMMØN /A/...

CØMMØN /A1/...

INTEGER P, P1.

```
      K1 = 0
      INI = 0
      M = 0
      EPS (2) = 1. E-5
      DØ  5  I = 1, N
      IVL (I) = 0
5     C (I) = 0
      Détermination de l'état de toutes les contraintes
      DØ 100      I = 1, NT
      DØ 10       J = 1, N1
10    T1 (J) = V (I, J)
      IF (ILU (I). EQ. 1) GØ TØ 25
      T2 (1) = 0
      DØ 20       J = 1, N
20    T2 (1) = T2 (1) + T1 (J) * X (J)
      Test si la contrainte φ_i est saturée.
      IF (ABS (T2 (1) - T1 (N1)) . LE . EPS (2)) GØ TØ 25
      IF (I. LE. P) GØ TØ 30
      GØ TØ 35
25    ILU (I) = 1
      Test si la contrainte φ_i est de type (a x_j + b = 0), dans ce cas x_j est
      variable de base.
      IF (IVX (I). EQ. 0) GØ TØ 90
      J = IVX (I)
      IF (IVL (J). LE. 0) K1 = K1 + 1
      IVL (J) = 1
      GØ TØ 90
30    INIT = 1
35    IF (T2(1) - T1 (N1)) 40, 100, 50
40    ILU (I) = -1                (contrainte non vérifiée négative)
      K = 1
      INI = 1
```

} si on ne veut pas garder en mémoire le tableau V des contraintes, on peut lire les contraintes dans un ordre séquentiel.

```
      GØ TØ 60
50    ILU (I) = 0
      K = 1
      IF (I. LE. P) K = -1
      IF (I. GE. P1)  GØ TØ 100
      Calcul de la fonction objectif auxiliaire.
60    DØ 70  J = 1, N
70    C (J) = C (J) + K * T1 (J)
      GØ TØ 100
90    M = M + 1     (incrémentation du nombre de contraintes saturées)
      DØ 95    J = 1, N  }  les coefficients de la contrainte saturée sont
95    G (M, J) = T1 (J) }  rangés dans G.
100   CØNTINUE
      Détermination des variables de base.
      IF (K1. EQ. M)  GØ TØ 120
      DØ 115       I = 1, M
      DØ 110       J = 1, N
      IF (IVL (J). EQ. 1) GØ TØ 110
      IF (G (I, J) . EQ. 0) GØ TØ 110     (Pour éviter que W soit une matrice
                                           singulière)
      IVL (J) = 1
      K1 = K1 + 1
      IF (K1 . EQ . M) GO TO 120
      GØ TØ 115
110   CØNTINUE
115   CØNTINUE
      DØ 117       I = 1, M
      J = N+1
116   J = J-1
      IF (J. LE. 0) GØ TØ 117
      IF (IVL (J). EQ. 1. ØR. G(I, J). EQ. 0) GØ TØ 116
      IVL (J) = 1
      K1 = K1 + 1
```

```
      IF (K1.EQ.M) GØ TØ 120
117   CØNTINUE
      IF (K1.EQ.M) GØ TØ 120        (Test si le nombre de variables de base
      INCI = 20                      est égal au nombre de contraintes
      RETURN                         saturées)
120   DØ 130    I = 1, M
      K = 0
      DØ 125    J = 1, N
      IF (IVL (J) .LE. 0) GØ TØ 125
      K = K + 1
      W (I, K) = G (I, J)        ⟵ Initialisation de la matrice W à inverser.
125   CØNTINUE
130   CØNTINUE
      RETURN
      END
```

INCI : Indicateur d'incident, il a la valeur :

0 : en cas de déroulement normal.

20 : contraintes redondantes.

5°) CALCUL DE LA DIRECTION DE MONTEE D. SOUS-PROGRAMMES GRADUI :

Ce sous-programme calcule la direction de montée à l'aide des formules (cf. I.2 : A.d) :

$$D_H(I) = f'_H(X^r) - f'_B(X^r) \cdot P^{-1} \cdot R$$

$$D_B(I) = -P^{-1} \cdot R \cdot D_H(I)$$

H : ensemble des indices des variables libres (IVL (I) = 0)

B : ensemble des indices des variables de base (IVL (I) = 1)

. La matrice P^{-1} est appelée dans le programme W (matrice inverse des coefficients des contraintes utiles G correspondant aux variables de base. (M × M).

. R est une sous-matrice de G correspondant aux variables libre dimensionnée à M × (N - M).

Paramètres d'entrée :

. W : matrice inverse (M × M)

. G : matrice (M × N - M) des coefficients des contraintes "utiles".

```
      SUBRØUTINE GRADUI
      CØMMØN /A/...
      K = 0
      DØ 10      I = 1, N
      IF (IVL (I) .EQ. 0) GØ TØ 10
      K = K + 1
      T1 (K) = C (I)      ←rangement dans T1 des coefficients ∂f/∂x_H
10    CØNTINUE
      DØ 20      J = 1, M
      T2 (J) = 0
      DØ 20      I = 1, M
20    T2 (J) = T2 (J) + W (I, J) * T1 (I)
      K = 0
      DØ 30      J = 1, N
      IF (IVL (J). EQ. 1) GØ TØ 30
      K = K+1
      T1 (K) = 0
      DØ 40      I = 1, M
40    T1 (K) = T1 (K) + T2 (I) * G (I, J)
      D (J) = C (J) - T1 (K)      ← Calcul de D_H correspondant aux
                                    variables libres.
      T1 (K) = D (J)
30    CØNTINUE
```

```
      Calcul de D_B correspondant aux variables de base . (D_B = -W.R.D_H)
      DØ 60      I = 1, M
      T2 (I) = 0
      K = 0
      DØ  50     J = 1, N
      IF (IVL (J). EQ.1) GØ TØ 50
      K = K+1
      T2 (I) = T2 (I) + T1 (K) * G (I, J)
50    CØNTINUE
60    CØNTINUE
      DØ 70      I = 1, M
      T1 (I) = 0
      DØ 70      J = 1, M
70    T1 (I) = T1 (I) + T2 (J) * W (I, J)
      K = 0
      DØ 80      I = 1, N
      IF (IVL (I). EQ. 0) GØ TØ 80
      K = K + 1
      D (I) = - T1 (K)
80    CØNTINUE
      RETURN
      END
```

Ce sous-programme est appelé par GRPROJ.

6°) SOUS-PROGRAMME GRPROJ TRADUISANT L'ALGORITHME DU GRADIENT

Ce programme traite deux phases de l'algorithme :

- la phase "saturation" si le nombre de contraintes saturées est inférieur strictement à N.

- La phase "d'optimisation" si le nombre de contraintes saturées est égal à N.

. Dans le ler cas, il se contente d'appeler le sous-programme GRADUI pour calculer la direction de montée D.

. Dans le second cas, il calcule les coefficients de Kuhn et Tucker à l'aide de la formule :

$$\lambda = P^{-1} \cdot f'(X_r)$$

Test : - si tous les coefficients λ_j correspondant aux contraintes unilatérales saturées sont négatifs ou nuls, il y a arrêt des itérations, l'optimum est obtenu.

- Si un des coefficients λ_j est strictement positif :
  - on libère la contrainte correspondante au plus grand $\lambda_j > 0$;
  - on fait évoluer la matrice W puisque le nombre N de contraintes saturées passe à N-1.

Paramètres d'entrée :

X : valeur du dernier itéré.

DW : matrice inverse des coefficients des contraintes correspondant aux variables de base.

C : coefficients de la fonction objectif.

```
      SUBROUTINE GRPRØJ
      CØMMØN /A/...
      CØMMØN /A1/
      INTEGER P, P1.
      IFI = 1
      Test si M < N        phase de "saturation"
      IF (M. LT. N) GØ TØ 130
      Phase d'optimisation M = N
10    DØ 20      J = 1, N
      T1 (J) = 0
      DØ 20      I = 1, N
20    T1 (J) = T1 (J) + C (I) * W (I, J)  ⟵ calcul des coefficients de Kuhn et
                                              Tucker.
```

```
      A = -0.01
      K = P1
      DØ 30      I = K, N      ← Test des Conditions de Kuhn et Tücker
      IF (T1 (I). LE. A) GØ TØ 30
      I1 = I
      A = T1 (I)
30    CØNTINUE
      IF (A. GT. 0) GØ TØ 40
      IFI = 0  ← (Tous les coefficients sont négatifs ou nuls, fin du pro-
      FX = 0          gramme)
      IF (INI. EQ. 1) RETURN  ← (Cas où le programme est en phase d'ini-
      DØ 25      I = 1, N          tialisation, il n'y a pas de solution réa-
25    FX = FX + X (I) * C (I)        lisable).
      Edition de résultats
      WRITE (6, 625) FX, ((I, X (I)), I = 1, N)
625   FØRMAT (20X, 19H MAXIMUM OBTENU  FX =, E10.4, /(30X, 1HX, I2,
                  3H = , E10.5))
      STØP 3
      Cas où un coefficient λ_j est positif, choix de la variable sortante et
      évolution de la matrice W.
40    K = 0
      DØ 50      I = 1, NT
      IF (ILU (I). NE. 1) GØ TØ 50
      K = K + 1
      IF (K. EQ. I1) GØ TØ 60
50    CØNTINUE
60    ILU (I) = 0  ← la contrainte d'indice I1 est libérée.
      Choix de la variable de base sortante.
      IF (IVX (J) . EQ. 0) GØ TØ 70
      J1 = IVX (I)  ← la contrainte libérée dépend de la seule variable
      GØ TØ 90          d'indice J1, on fait sortir cette variable de la base
70    DØ 80      I = 1, N
      J1 = I
```

```
      IF (IVL (I) .EQ.0) GØ TØ 80
      DØ 75  J = P1, NT
      IF (ILU (J).NE.1)  GØ TØ 75
      IF (IVX (J).NE.I) GØ TØ 75
      GØ TØ 80
75    CØNTINUE
      IF (ABS (W(J1, I1)).GT.EPS (1))  GØ TØ 90
80    CØNTINUE
      INCI = 30   ⟵ Cas où il n'est pas possible de libérer une variable
      RETURN
90    IVL (J1) = 0
      M = M - 1   ⟵ Le nombre de variables de base est décrémenté de 1.
      Evolution de la matrice W qui passe de rang M à M-1.
      DØ 100      J = 1, N
      IF (J.EQ.I1) GØ TØ 100
      DØ 95       I = 1, N
      IF (I.EQ.J1) GØ TØ 95
      W (I, J) = W (I, J) - (W (J1, J) * W (I, I1)/W (J1, I1))
95    CØNTINUE
100   CØNTINUE
      Suppression de la ligne J1 et la colonne I1 de zéros dans W.
      IK = 0
      DØ 110      I = 1, N
      IF (I.EQ.J1) GØ TØ 110
      IK = IK + 1
      JK = 0
      DØ 105      J = 1, N
      IF (J.EQ.I1) GØ TØ 105
      JK = JK + 1
      W (IK, JK) = W (I, J)
105   CØNTINUE
110   CØNTINUE
```

```
        Suppression de la ligne Il correspondant à la contrainte libérée
        dans G.
        K = 0
        DØ 120      I = 1, N
        IF (I. EQ. Il) GØ TØ 120
        K = K + 1
        DØ 115      J = 1, N
115     G (K, J) = G (I, J)
120     CØNTINUE
130     CALL GRADUI  ⟵ Calcul de la direction de montée.
        RETURN
        END
```

. INCI : Indicateur d'incident prend la valeur :

- 0 en cas de déroulement normal.
- 30 impossibilité de libérer une variable, contraintes redondantes.

. Sous-programme appelé : GRADUI

7°) CALCUL DU PAS H - SATURATION D'UNE CONTRAINTE - SATUR :

Ce sous-programme a pour rôle de calculer le pas h pour le prochain itéré $X_{r+1} = X_r + h \cdot D$ (cf. II.A.c.)

$$h = \min_{h \in K} \left(\frac{\delta_k}{A'_k \cdot D} \right)$$

K : ensemble des indices des contraintes unilatérales non saturées.

. Lors de la saturation d'une contrainte, le programme fait évoluer les matrices W et G.

. INCI = Indicateur d'incident prend la valeur :

0 en cas de déroulement normal.

40 pas h infini (solutions infinies).

41 choix d'une variable entrante impossible (contraintes redondantes).

Paramètres d'entrée :

X : valeur du dernier itéré X_r.

D : direction de montée.

W, G : matrices des coefficients des contraintes utiles.

V : tableau contenant l'information de toutes les contraintes. Si l'utilisateur ne veut pas le mémoriser, il faut lire les coefficients des contraintes sur mémoire périphérique.

SUBROUTINE SATUR

```
      CØMMØN /A/...
      CØMMØN /A1/...
      INTEGER P, P1
      ITER = ITER + 1      (Incrémentation du nombre d'itérations puisque
                            ce programme calcule le nouveau point X_r).
      H1 = 1.E+30
      U1 = 0
      Calcul du pas h, saturation d'une contrainte.
      DØ 50      I = 1, NT
      IF (ILU (I). EQ. 1) GØ TØ 50
      J = 1
      IF (ILU (I) . GE. 0) GØ TØ 10
      J = -1
10    DØ 20   K = 1, N1
      IP (K) = 0
20    T1 (K) = J * V (I, K) <-- (Si V non mémorisé, lecture sur mémoire
                                 externe)
      E1 = 0
      E2 = 0
```

```
      DØ 30       J = 1, N
      E1 = E1 + T1 (J) * X (J)
30    E2 = E2 + T1 (J) * D (J)
      E1 = T1 (N1) -E1
      IF (E2.GE. 0.ØR. E1. GE. 0) GØ TØ 50
      U1 = 1   <-- Indicateur qu'une contrainte au moins a été saturée
      H = E1/E2
      IF (H. GT. H1)  GØTØ 50   <-- On choisit H = Min H_i
      I1 = I
      DØ 40       K = 1, N1
40    T2 (K) = T1 (K)
      H1 = H
50    CØNTINUE
      H = H1
      IF (U1. GE. 1) GØ TØ 60
      INCI = 40   <-- Incident : aucune contrainte a été saturée, pas h ∞.
      RETURN
60    ILU (I1) = 1
      M = M + 1   <-- Incrémentation du nombre de contraintes saturées.
      Calcul de la fonction F (X_r+1)
      FX = 0
      DØ 70       I = 1, N
      X (I) = X (I) + D (I) * H
70    FX = FX + X (I) * C (I)
      Edition du nombre d'itérations et de la valeur de la fonction f (X_r+1)
      WRITE (6, 670) ITER, FX
670   FORMAT (40X, I2, 8X, E10.2)
      IF (INI. NE. 0) RETURN
      Recherche d'une variable entrante.
      J1 = IVX (I1)
      IF (J1. LE. 0) GØ TØ 80
      IF (IVL (J1) .EQ. 1) GØ TØ 80
```

```
      IVL (J1) = 1          ← Cas où la contrainte saturée dépend de la seule
      GØ TØ 100               variable x_j1 et que celle-ci est libre.
80    DØ 90      I = 1, N
      IF (IVL (I). EQ. 1) GØ TØ 90
      IF (M. EQ. 1. AND. ASS (T2 (I)). LT. EPS(1)) GØ TØ 90
      IVL (I) = 1           ← On fait entrer dans la base la première variable
      J1 = I                  libre rencontrée.
      GØ TØ 100
90    CØNTINUE
100   K = 0
      IP (J1) = 1
      W (M, M) = T2 (J1)
      DØ 110     J = 1, N
      G (M, J) = T2 (J)     ← mémorisation des coefficients de la nouvelle
      T1 (J) = G (J, J1)      contrainte saturée.
      IF (IVL (J). EQ. 0. ØR. J. EQ. J1) GØ TØ 110
      K = K+1
      T2 (K) = T2 (J)
110   CØNTINUE
      IF (M. GT. 1) GO TO 120
      W (M, M) = 1/ W (M, M)  ← cas où M = 1, l'inverse de W est évident
      RETURN
120   CØNTINUE
      Calcul de la matrice W_r+1 (M, M) en fonction de W_r (M-1, M-1)
      M1 = M - 1
      DØ 130     I = 1, M1
      W (I, M) = 0
      DØ 130     J = 1, M1
130   W (I, M) = W (I, M) + W (I, J) * T1 (J)
      E = 0
      DØ 140     I = 1, M1
140   E = E + W (I, M) * T2 (I)
```

```
      E = W (M, M) -E
      Si le scalaire E est nul, le choix de la variable entrante est mauvais,
      on recherche une autre variable.
      IF (ABS (E) .GT. EPS (1)) GØ TØ 150
      IVL (J1) = 0
      DØ 145      I = 1, N
      IF (IP (I). EQ.1. ØR. IVL (I). EQ.1) GØ TØ 145
      J1 = I
      IVL (J1) = 1
      GØ TØ 100
145   CØNTINUE
      INCI = 41   <-- Incident : aucune variable libre ne peut entrer dans la
                                  base.
      RETURN
150   E = 1/E
      DØ 160      J = 1, M1
      W (J, M) = - W (J, M) * E
      W (M, J) = 0
      DØ 155      I = 1, M1
155   W (M, J) = W (M, J) - T2 (I) * W (I, J) * E
160   CØNTINUE
      W (M, M) = E
      DØ 170      I = 1, M1
      DØ 170      J = 1, M1
170   W (I, J) = W (I, J) + W (I, M) * W (M, J) /E
      IF (K1. EQ. M) GØ TØ 185   <-- Test : si l'indice de la variable entrante
                                     est égal à M, pas de modification
      DØ 180      J = 1, M           sur les colonnes.
      T2 (J) = W (K1, J)
      W (K1, J) = W (M, J)
      K = K1 + 1
      DØ 175      J = K, M
      T1 (1) = W (I, J)
```

```
      W (I, J) = T2 (J)
175   T2 (J) = T1 (1)
180   CØNTINUE
185   K = 0
      DØ 190     I = 1, NT
      IF (ILU (I) . EQ. 0) GØ TØ 190
      IF (I. EQ. I1) GØ TØ 195
      K = K + 1
190   CØNTINUE
195   K1 = K + 1
      IF (K1. EQ. M) RETURN  <-- Test : si l'indice de la contrainte saturée
                                 est égal à M, pas de permutation
      DØ 200     J = 1, N        sur les lignes (M et I1) et colonnes
      T1 (1) = G (M, J)
      T2 (1) = W (J, M)
      DØ 200     I = K1, M
      T2 (2) = W (J, I)
      W (J, I) = T2 (1)
      T2 (1) = T2 (2)
      T1 (2) = G (I, J)
      G (I, J) = T1 (1)
 95   T1 (1) = T1 (2)
      RETURN
      END
```

8°) APPLICATION A L'APPROXIMATION D'UNE FONCTION AU SENS DE TCHEBYCHEFF :

L'approximation d'une fonction quelconque que l'on propose de faire est une approximation au sens de Tchebycheff sur un nombre discret de points.

Définition du problème :

. Soit f (x) une fonction quelconque et un ensemble fini de points $\{x_0, x_1, \ldots, x_n\}$.

On veut approcher f (x) par une fonction P (x) :

$$P(x) = a_0 u_0(x) + a_1 u_1(x) + \ldots + a_p u_p(x)$$

où : - $u_i(x)$ sont des fonctions données, par exemple des polynômes (polynômes de Tchebycheff) ou des fonctions de type trigonométrique. Elles peuvent être quelconques.

- a_i sont des scalaires que l'on veut calculer de manière à avoir la meilleure approximation de la fonction f (x).

Le problème est de minimiser la norme $\|f(x) - P(x)\|$.

$$\|f(x) - P(x)\| = \max_i |\varepsilon_i| \quad \forall i \in \{1, \ldots, n\}$$

ε_i est tel que :

$$\forall x_i \qquad \varepsilon_i = f(x_i) - a_0 u_0(x_i) - a_1 u_1(x_i) \ldots\ldots - a_p u_p(x_i).$$

Posons z variable auxiliaire $z = \|f(x) - P(x)\|$.

Le problème est de minimiser z ou maximiser -z,

tel que : $$|\varepsilon_i| < z \qquad \forall i \{1, \ldots n\}. \qquad (1)$$

Si on supprime la valeur absolue (1) peut s'écrire :

$$\begin{cases} \varepsilon_i - z < 0 \\ \varepsilon_i + z > 0. \end{cases}$$

Le problème est un problème de programmation linéaire :

$$\text{Max } -z$$

$$\begin{cases} f(x_i) - (a_0 u_0(x_i) + a_1 u_1(x_i) + \ldots + a_p u_p(x_i)) - z \leq 0 \\ f(x_i) - (a_0 u_0(x_i) + a_1 u_1(x_i) + \ldots + a_p u_p(x_i)) - z \geq 0 \end{cases} \qquad (2)$$

$$\forall i \in \{1, \ldots n\}.$$

(2) peut s'écrire aussi :

$$\begin{cases} z + \sum_{j=0}^{P} a_j u_j (x_i) \geq f(x_i) \\ z - \sum_{j=0}^{P} a_j u_j (x_i) \geq - f(x_i) \end{cases}$$

$$\forall i \in \{1, n\}.$$

. Le nombre de variables est égal à p + 2.

. Si le nombre de points choisis est égal à n, le nombre de contraintes est égal à 2n.

Le nombre de contraintes doit être supérieur au nombre de variables sinon, on a un système indéterminé, on doit avoir : $2n > p + 2$.

Exemple : Approximation de la fonction sin x par les polynômes de Tchebycheff

les polynômes de Tchebycheff se définissent par récurrence

. $u_0 (X) = 1$

. $u_1 (X) = X$

. $u_{i+1} (X) = 2 * X * u_{i-1} (X) - u_{i-2} (X)$ $\qquad 2 \leq i \leq p.$

On se fixe p = 4.

On choisit 10 points sur l'intervalle [-1, +1], le nombre de contraintes est égal à 20.

Le problème est :

Max - z

$$\begin{cases} z + \sum_{j=0}^{4} a_j u_j (x_i) \geq \sin (x_i) & \forall i \in \{1, \ldots, 10\} \\ z - \sum_{j=0}^{4} a_j u_j (x_i) \geq - \sin (x_i). \end{cases}$$

Résultats obtenus par programme :

On obtient pour écart maximum : $0.294.10^{-3}$.

Les coefficients a_i sont :

$$a_0 = -0.479.10^{-3}$$

$$a_1 = 0.879$$

$$a_2 = -0.823.10^{-3}$$

$$a_3 = -0.398.10^{-1}$$

$$a_4 = -0.496.10^{-3}$$

On choisit 20 points x_i équi-répartis sur l'intervalle [-1, +1] et on calcule les écarts $\varepsilon_i = P(x_i) - \sin(x_i)$.

On obtient les résultats suivants :

x_i	$10^{-5}.\ \varepsilon_i$	x_i	$10^{-5}.\ \varepsilon_i$
-1	-29.07	10^{-8}	15.19
-0.9	21.52	0.1	26.31
-0.8	29.11	0.2	29.10
-0.7	15.21	0.3	21.78
-0.6	-4.89	0.4	5.47
-0.5	-21.37	0.5	-14.78
-0.4	-29.10	0.6	-29.11
-0.3	-26.78	0.7	-21.88
-0.2	-16.03	0.8	29.09
-0.1	- 4.83	0.9	15.34

Les résultats sont représentés graphiquement sur la figure I.

-1
-0.9
-0.8
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
$\bar{z} = 0.284 \cdot 10^{-3}$

9°) AVANTAGES ET INCONVENIENTS DE LA METHODE DU GRADIENT PAR RAPPORT AU SIMPLEX :

Les avantages et les inconvénients d'une méthode par rapport à l'autre sont déduits des résultats obtenus sur les exemples traités dans ce travail.

En ce qui concerne la place mémoire occupée par le programme utilisant la Méthode du Gradient, celui-ci a été conçu de manière à occuper le minimum de place sans utiliser des techniques de mémorisation sophistiquée (matrice sans trou).

a) Inconvénients :

. Le programme utilisant la méthode du Gradient est plus long qu'un programme de Simplex. Ce programme a été conçu de manière à éviter l'inversion de la matrice des coefficients des contraintes "utiles" à chaque itération. Il peut être simplifié si l'on inverse la matrice à chaque itération.

. Si un problème est de forme "standard" c'est-à-dire du type :

$$\begin{cases} \text{Max } f(X) \\ AX = E \\ X >> 0. \end{cases}$$

L'algorithme du Simplex est certainement plus performant que la méthode du Gradient et il parait préférable d'utiliser le Simplex.

b) Avantages :

. L'algorithme du Gradient admet comme solution de départ n'importe quel point de l'espace $\mathbb{R}^n$ dans lequel on se place. De ce fait, si pour un problème particulier, on connait une solution réalisable, ou un point proche d'une solution réalisable, on peut gagner des itérations correspondant à la phase "initialisation".

. Quelque soit la forme "standard", "canonique" ou "mixte" sous laquelle se présente le problème de programmation linéaire, le nombre de variables réelles définies au départ reste inchangé. Ce qui est un réel avantage par rapport au Simplex qui doit transformer tout problème sous forme standard. Pour des contraintes de type unilatérale ($\varphi_i = A_i X \geq e_i$) le Simplex doit transformer cette contrainte en contrainte bilatérale, il doit ajouter une variable d'écart et la contrainte de signe associée à cette variable.

$A_i X \geq e_i$ devient : $A_i X + X_e = e_i$ avec $X_e \geq 0$.

. Pour trouver une solution réalisable, l'algorithme du Simplex génère des variables artificielles, ce qui n'est pas le cas dans l'algorithme du Gradient.

. Considérons un problème de type "canonique"

$$\text{Max } f(X)$$
$$\begin{cases} AX >> E \\ X >> 0 \end{cases} \qquad X \in \mathbb{R}^n$$

. N variables réelles

. M contraintes (exclu les contraintes de signe).

Dans le Simplex, le nombre de variables passe de N à N + M puisqu'il faut ajouter M variables d'écart pour transformer le problème sous forme "standard".

. Il faut signaler que les contraintes de signe sont considérées dans la Méthode du Gradient comme n'importe quelle autre contrainte de type unilatérale, alors que, dans ce cas le Simplex doit introduire des contraintes et des variables supplémentaires pour se ramener au cas standard et faire apparaître des contraintes de signe. Pour un problème tel que celui de l'approximation d'une fonction vu dans le paragraphe précédent (I.8), si on utilise le Simplex, on introduit les contraintes et variables supplémentaires suivantes :

Pour chaque variable réelle X_i, on pose :

$$X_i = X_{i_1} - X_{i_2} \geq 0$$

$$X_{i_1} \geq 0$$

$$X_{i_2} \geq 0$$

. Pour la méthode du Gradient, il n'y a aucun ajout de contraintes ou de variables supplémentaires. Ce qui au point de vue place mémoire peut représenter un gain non négligeable.

Si tous les coefficients des contraintes sont sur mémoire périphérique, dans le programme du Gradient les tableaux les plus grands sont deux matrices carrées W et G dimensionnées au nombre de variables N. Si le rapport N/NT du nombre de variable N sur le nombre de contraintes NT est faible (jusqu'à 0.5), le programme peut traiter des problèmes à grand nombre de contraintes.

c) Remarques sur l'utilisation de la Méthode du Gradient ou du Simplex :

- Si le problème de programmation linéaire se présente au départ sous forme "standard", il parait préférable d'utiliser le Simplex.

- Si le problème traité est sous forme "canonique" et le nombre de contraintes NT est grand (NT $\geq 2 * N$) par rapport au nombre de variables N, il semble naturel d'utiliser la Méthode du Gradient.

- Si le problème est de type "mixte" :

$$\text{Max } f(X)$$

$$\begin{cases} AX = E \\ A'X >> E' \\ X >> 0 \end{cases} \qquad X \in \mathbb{R}^n$$

si les contraintes de type unilatéral sont peu nombreuses par rapport aux contraintes bilatérales on peut utiliser le Simplex ; dans le cas contraire il vaudra mieux utiliser l'algorithme du Gradient.

- Dans le cas où il n'y a pas de contraintes de signe comme dans l'exemple traité dans le paragraphe précédent, il est vivement conseillé d'utiliser la Méthode du Gradient.

- Pour des problèmes dont on connait une solution réalisable, ou pour des problèmes où l'on est amené à modifier une contrainte, ou un coefficient de la fonction objectif, ou problème initial dont on connait la solution optimale, il est normal d'utiliser la méthode du Gradient qui permet d'introduire comme solution de départ pour le nouveau problème l'ancienne solution optimale. On gagne un nombre d'itérations et par conséquent du temps machine.

CHAPITRE IV

METHODES ET ALGORITHMES DANS LE CAS OU LA FONCTION OBJECTIF EST QUADRATIQUE

1°) NOTATIONS :

a) Soit $X \in \mathbb{R}^n$, trouver $X^* (x_1^*, x_2^*, \ldots, x_n^*)$ qui rend f (X) optimum (maximum ou minimum suivant le cas étudié).

. f (X) est une fonction quadratique de X.

. X doit vérifier les contraintes linéaires suivantes :

$$\varphi_\ell (X) = 0 \qquad \forall \ell \in L$$

$$\varphi_m (X) \geq 0 \qquad \forall m \in M$$

L : ensemble des indices des contraintes bilatérales.

M : ensemble des indices des contraintes unilatérales.

. Soit p le nombre de contraintes bilatérales et q le nombre de contraintes unilatérales.

b) Notations sous forme matricielle :

Soit $X \in \mathbb{R}^n$, trouver X tel que :

$$\text{Max } Z = -\frac{1}{2} \, {}^tX \, . \, C \, . \, X + {}^tu \, . \, X.$$

Avec les contraintes suivantes :

$$\begin{cases} A \, . \, X = E & : \ p \text{ contraintes bilatérales} \\ A' \, . \, X >> E' & : \ q \text{ contraintes unilatérales.} \end{cases}$$

. On suppose que la matrice carrée C d'ordre n est symétrique et positive semi-définie.

. ${}^t u$: un vecteur de scalaires à n composantes.

. A : matrice $(p \times n)$ des coefficients des contraintes bilatérales.

. A' : matrice $(q \times n)$ des coefficients des contraintes unilatérales.

. E : vecteur colonne à p composantes.

. E' : vecteur colonne à q composantes.

c) Remarques :

. Le cas mixte est une généralisation du cas "standart" (problème avec contraintes bilatérales et contraintes de signe) et du cas "canonique" (problèmes avec contraintes unilatérales).

. On étudie uniquement le cas des problèmes de maximisation. Si l'on doit minimiser une fonction Z on se ramène au cas précédent en cherchant Max (-Z).

. On suppose que :

- aucune contrainte n'est redondante ;
- toutes les contraintes sont compatibles.

2°) DIFFERENCES ET POINTS COMMUNS ENTRE LES PROBLEMES DE PROGRAMMATION LINEAIRE ET QUADRATIQUE

a) Les contraintes sont linéaires dans les deux cas, ce qui a pour conséquence : la recherche d'une solution réalisable est identique dans les deux types de problèmes. Une solution réalisable est un vecteur X de $\mathbb{R}^n$ qui vérifie l'ensemble des contraintes du problème.

On a vu dans le chapitre II.3.a que la recherche d'une solution réalisable ne fait pas intervenir la fonction objectif du problème initial. Donc toutes les méthodes utilisées pour la phase "initialisation d'un problème de programmation linéaire sont identiques à celles utilisées pour la recherche d'une solution de base d'un problème de programmation quadratique.

b) Dans les problèmes à fonction objectif linéaire, la solution optimale n'est autre qu'un sommet du polyèdre saturant n contraintes. Au contraire, une fonction objectif quadratique peut atteindre son optimum sur une arête ou à l'intérieur du polyèdre ; on aura au plus n contraintes saturées. D'où une caractérisation de l'optimum différente dans le cas quadratique que nous exposerons dans la méthode de résolution.

3°) ALGORITHME DU GRADIENT PROJETE :

Comme dans le cas linéaire, on a une solution réalisable $\bar{X}$ et on recherche une solution optimale. Le nombre de contraintes saturées (bilatérales vérifiées + unilatérales saturées) est au plus égal à n.

a) Caractérisation de l'optimum :

Comme dans le cas linéaire, on utilise les conditions de Kuhn et Tucker (cf. chap. II.2.B.a) que nous ne rappellerons pas. Le calcul des coefficients est différent du fait que dans le cas linéaire on a exactement n contraintes saturées alors que dans notre cas on a au plus n contraintes saturées.

L'équation définissant les coefficients est :

$$(1) \qquad \frac{\partial f}{\partial x_i}(\bar{X}) - \sum_{\ell \in L} \frac{\partial \varphi_\ell}{\partial x_i}(\bar{X})\,\lambda_\ell - \sum_{k \in K} \frac{\partial \varphi_k}{\partial x_i}(\bar{X})\,\mu_k = 0$$

$$i = 1, 2, \ldots, n$$

L : ensemble des indices des contraintes bilatérales.

K : ensemble des indices des contraintes unilatérales saturées.

Sous forme matricielle l'équation (1) s'écrit :

$$(2) \quad C = {}^tP \,.\, \Lambda \qquad \text{où} \qquad \Lambda = \begin{pmatrix} \lambda_\ell \\ \mu_k \end{pmatrix}$$

C : gradient de la fonction objectif au point $\vec{X}$, vecteur à n composantes.

${}^tP$: matrice rectangulaire des coefficients des $\underline{m}$ ($m \leq n$) contraintes "utiles" (bilatérales vérifiées + unilatérales saturées) dimensionnée à ($n \times m$) correspondant aux m variables de base.

Λ : m coefficients de Kuhn et Tucker.

Remarque : Dans le cas linéaire, ${}^tP$ est une matrice carrée que l'on peut inverser.

. Pour résoudre le système (2), on utilise la méthode de Golub (multiplication par des matrices de Householder qui conservent la norme euclidienne et diagonalisent la matrice des coefficients des inconnues : cf. Méthodes et Techniques de l'Analyse Numérique. J. LEGRAS - Chez Dunod).

(Pour la programmation de cette méthode de résolution du système (2), voir le sous-programme MCADIR de bibliothèque).

Connaissant les coefficients de Kuhn et Tucker μ_k ($k \in K$), on teste leur signe ; deux cas peuvent se présenter.

1er cas : Tous les coefficients sont négatifs ou nuls, alors on calcule la direction de montée D et on teste la norme D. Si toutes les composantes d_i de D sont inférieures à un ε en valeur absolue, l'optimum est atteint, l'algorithme est fini.

2ème cas : Si le plus grand des coefficients est positif, on libère ou on désature la contrainte associée à ce coefficient et on calcule la direction de montée D.

Remarque : La caractérisation de l'optimum est différente du cas linéaire. En plus du test des coefficients de Kuhn et Tucker, on teste la norme de la direction D.

b) Calcul de la direction de montée D :

On se propose de chercher une direction D de norme euclidienne δ, telle que pour h positif fixé, $f(X^r + h\,D)$ soit maximum, et de plus D doit être direction admissible c'est-à-dire :

$$\varphi_\ell (X^r + h\,D) = 0 \qquad \ell \in L \quad (j \in J = L \cup K_1)$$

$$\varphi_{k_1} (X^r + h\,D) \geq 0 \qquad k_1 \in K_1$$

L : ensemble des contraintes bilatérales.

K_1 : ensemble des contraintes unilatérales vérifiées saturées.

Pour simplifier ce problème, nous cherchons une meilleure direction locale, ce qui signifie que nous considérons h comme "petit" et linéarisons la fonction et les contraintes. On obtient :

$$\max \Omega = D \cdot f'(X_r)$$

avec $$\psi_o = \left(\sum_{j=1}^{n} d_j^2\right) - \delta^2 = 0$$

$$\psi_\ell = D \cdot \varphi'_\ell (X_r) = 0 \qquad \forall \ell \in L$$

$$\psi_{k_1} = D \cdot \varphi'_{k_1} (X_r) \geq 0 \qquad \forall k_1 \in K_1.$$

Rappelons que D est le tableau d'éléments $d_1, d_2, \ldots, d_n$ inconnues de notre problème et que :

$$D \cdot f'(X_r) = \sum_{i=1}^{n} d_i \cdot \frac{\partial f(X_r)}{\partial x_i}$$

$$D \cdot \varphi_j'(X_r) = \sum_{i=1}^{n} d_i \cdot \frac{\partial \varphi_j(X_r)}{\partial x_i} \qquad \forall j \in J.$$

Nous nous contentons de rechercher si la direction de meilleure montée, obtenue en maintenant saturée pendant l'étape d'origine X_r, les contraintes saturées en X_r, est satisfaisante (les coefficients de Kuhn et Tucker associés à ce problème doivent alors être tous négatifs ou nuls) ; dans le cas contraire, de chercher une direction de montée admissible, sous la condition restrictive de ne libérer qu'une seule des contraintes saturées en X_r.

Supposons d'abord que toutes les contraintes d'indice $j \in J_r$ sont maintenues saturées. Les éléments d_i du tableau D et les multiplicateurs λ_o, λ_ℓ et λ_{k_1} relatifs à notre problème vérifient les équations (1) du paragraphe précédent qui deviennent :

$$(3) \qquad \Omega' = \lambda_o \cdot \psi_o' + \sum_{\ell \in L} \lambda_\ell \, \psi_\ell' + \sum_{k_1 \in K_1} \lambda_{k_1} \cdot \psi_{k_1}'$$

La linéarité de Ω, des ψ_j et ψ_{k_1} par rapport aux variables d_i entraîne :

$$\Omega' = f'(X_r) \qquad \text{car} \qquad \frac{\partial \Omega}{\partial d_{i'}} = \frac{\partial f(X_r)}{\partial x_i} \qquad i' = 1, 2, \ldots, n$$

$$\psi_i' = \varphi_i'(X_r) \qquad \text{car} \qquad \frac{\partial \psi_i}{\partial d_i} = \frac{\partial \varphi_i(X_r)}{\partial x_i} \qquad \forall i \in L \text{ ou } K_1$$

$$\psi_o' = 2 \cdot D.$$

La relation (3) devient donc :

$$f'(X_r) = 2 \lambda_o D + \sum_{\ell \in L} \lambda_\ell \, \varphi_\ell'(X_r) + \sum_{k_1 \in K_1} \lambda_{k_1} \cdot \varphi_{k_1}'(X_r)$$

que nous transformons sous la forme :

$$2\lambda_o D = f'(X_r) - \sum_{\ell \in L} \lambda_\ell \cdot \varphi'_\ell(X_r) - \sum_{k_1 \in K_1} \lambda_{k_1} \cdot \varphi'_{k_1}(X_r)$$

λ_o est un coefficient de proportionnalité et nous pouvons toujours supposer que nous avons choisi δ pour que $2\lambda_o = 1$.

La relation (3) s'écrit donc :

$$\boxed{D = f'(X_r) - \sum_{i \in J} \lambda_i \cdot \varphi'_i(X_r)} \qquad (4)$$

J : ensemble des indices des contraintes bilatérales et unilatérales saturées.

Le tableau D doit vérifier les contraintes.

$$D \cdot \varphi'_i = 0 \qquad \forall i \in J \qquad (5)$$

Substituons D défini par (4) et remplaçons son expression dans l'équation (5), on obtient :

$$\left(\sum_i \lambda_i \varphi'_i(X_r)\right) \cdot \varphi'_{i'}(X_r) = f'(X_r) \cdot \varphi'_{i'}(X_r) . \qquad \forall i' \in J$$

On pose :

$$[\Phi'] = \left(\frac{\partial \varphi_i(X_r)}{\partial x_j}\right) \qquad \begin{matrix} \forall i \in J \\ \forall j \in \{1, 2, \ldots, n\} \end{matrix}$$

$$B = [\Phi'] \cdot {}^t[\Phi']$$

$$F = {}^t[f'(X_r)] \cdot {}^t[\Phi'] .$$

$[\Phi']$: matrice des dérivées des contraintes dites "utiles" (bilatérales + unilatérales saturées).

Le système définissant les λ_i se formalise en :

$$[B] \cdot [\Lambda] = [E] .$$

Une fois les λ_i calculés, la relation (4) définit D, direction de meilleure montée locale si nous maintenons saturées toutes les contraintes précédemment saturées en X_r.

Si nous libérons une contrainte dont le coefficient de Kuhn et Tucker est positif, D reste direction de montée admissible.

Sous forme matricielle, la relation (4) s'écrit :

$$D = [f'(X_r)] - {}^t[\Phi'] \cdot [\Lambda] .$$

Remarque : Contrairement à la méthode du gradient Réduit "étendu" utilisée dans le cas linéaire où la direction de montée D peut ne pas être compatible avec la liaison libérée, avec le gradient projeté, D reste direction de montée admissible.

Dans le cas où il y a (n-1) contraintes saturées et donc une seule variable libre, les deux méthodes sont identiques.

c) Calcul du pas h :

Comme nous l'avons indiqué dans la première partie (I.4.a), l'optimum s'il existe en programmation quadratique peut se trouver à l'intérieur ou sur la frontière du domaine réalisable.

Soit X_r le point obtenu à la rième itération et D_r la direction de montée, le pas h permet de déterminer le point X_{r+1} ($X_{r+1} = X_r + D_r h$). La fonction objectif est fonction du second degré en h.

$$f(X_r + D_r h) = -\frac{1}{2} {}^t(X_r + D_r h) \cdot C \cdot (X_r + D_r h) + {}^t p \cdot (X_r + D_r h).$$

Pour calculer h, il suffit de résoudre l'équation du ler degré en h :

$$\frac{df}{dh} \quad (X_r + D_r \cdot h) = 0 \qquad (5)$$

L'équation (5) se développe en

$$-{}^t D_r \cdot C \cdot (X_r + D_r\, h) + {}^t p \cdot D_r = 0$$

$$({}^t D_r \cdot C \cdot D_r)\, h = - {}^t D_r\, C\, X_r + {}^t p\, D_r .$$

Si la direction D_r n'est pas nulle, h est égal à :

$$h^* = \frac{- {}^t D_r \cdot C \cdot X_r + {}^t p\, D_r}{{}^t D_r \cdot C \cdot D_r} .$$

Cet accroissement h^* détermine sur la droite D_r un point X^*_{r+1} qui doit être réalisable. Par conséquent, il faut vérifier si X^*_{r+1} est réalisable, pour cela on calcule comme dans le cas linéaire (cf. II. 2. A) le pas h_i qui détermine un point X_{r+1} réalisable. On rappelle la formule qui permet le calcul de h_i.

$$h_i = \frac{\delta_i}{A'_i \cdot D_r} \qquad \text{et} \qquad \delta_i = E'_i - A'_i \cdot D_r$$

$$h_o = \underset{i \in K_1}{\text{Min}} \; (h_i).$$

K1 : ensemble des indices des contraintes unilatérales non saturées.

Si h^* est inférieur à l'accroissement h_o, on choisit h^* comme pas puisqu'il détermine un point X_{r+1} réalisable qui améliore la valeur de la fonction objectif.

Donc : $\underline{h = \text{Min}\,(h^*, h_o)}$

Le calcul du pas h se fait de manière exacte.

4°) METHODE DE RESOLUTION D'UN PROBLEME DE PROGRAMMATION QUADRATIQUE PAR LA METHODE DU GRADIENT :

Soit (P) un programme quadratique sous forme mixte :

$$(P) \begin{cases} A \,.\, X = B \\ A' \,.\, X >> B' \\ \text{Max } -\frac{1}{2}\ {}^{t}X \,.\, C \,.\, X + {}^{t}p \,.\, X. \end{cases}$$

On distingue deux phases dans la méthode de résolution :

- une phase initialisation :

A partir d'un vecteur $\bar{\xi}$ quelconque, déterminer un vecteur $\bar{X}$ réalisable. Cette phase est identique au cas linéaire (cf. II.3) puisque la fonction objectif n'intervient pas.

- une phase d'optimisation :

A partir d'une solution $\bar{X}$ réalisable qui sature q contraintes ($q \leq n$) déterminer la solution optimale X^* si elle existe.

On applique les conditions de Kuhn et Tucker, deux cas peuvent se présenter :

1er cas : si les conditions sont vérifiées (tous les coefficients λ_i sont négatifs ou nuls), on calcule la direction de montée D :

i) si la direction D est nulle, l'optimum est obtenu.

ii) Dans le cas contraire, on détermine un nouveau point X sur lequel on applique les conditions de Kuhn et Tucker.

2ème cas : Les conditions ne sont pas vérifiées ; on libère la contrainte correspondant au plus grand coefficient de Kuhn et Tucker.

On calcule de nouveau les coefficients λ_i et ensuite la direction de montée D qui nous permet de déterminer un nouveau point.

Remarque : L'évolution de la matrice P^{-1} (matrice inverse des contraintes "utiles" correspondant aux variables de base) est la même que dans le cas linéaire.

5°) ACCELERATION DE CONVERGENCE - METHODE DES DIRECTIONS CONJUGUEES :

a) Caractéristiques de la méthode :

La méthode des directions conjuguées est une méthode classique de montée, calculée à partir du gradient, et procédant par itérations successives. Nous nous contentons de rappeler le principe de la méthode et les principaux résultats sans démonstration.

On utilise cette méthode dans la phase "optimisation", lorsqu'il n'y a pas saturation d'une nouvelle contrainte, le pas h étant égal à h^*. (cf. IV.3.c).

Les itérations sont groupées en cycles, la caractéristique d'un cycle étant que les contraintes maintenues saturées à la première étape du cycle resteront saturées pendant tout le cycle, ceci par convention.

Un cycle sera terminé :

- soit lorsqu'une nouvelle contrainte est saturée au cours d'une étape, dans ce cas il n'y aura pas accélération de convergence.
- soit lorsque le nombre d'étapes du cycle est égal à s (s égal au nombre de variables hors base).

Chaque cycle est précédé d'une étape préliminaire (non comprise dans les s étapes du cycle) ou l'on teste si les contraintes saturées en début de cycle

doivent rester ou non saturées à l'étape suivante.

A chaque étape, à partir de $\overline{X}_r$ nous calculons la direction de montée D_r dans l'espace des s variables hors base. Pour maintenir saturées les contraintes précédemment saturées en début de cycle, nous faisons une élimination simple ($D_B = f(D_H)$) utilisée dans la méthode du gradient Réduit "étendu" (cf. II.2.).

Nous déterminons ensuite $\overline{X}_{r+1}$ sur la direction D_r en calculant le pas h. Si ce dernier n'est pas égal à h^*, l'usage de la méthode des directions conjuguées ne parait pas utile. Dans ce cas nous prendrons $\overline{X}_{r+1}$ comme vecteur initial d'une nouvelle étape préliminaire.

Dans le cas contraire, nous poursuivons le cycle pendant s étapes (s : nombre de variables hors base, nombre constant par convention pendant un cycle).

b) Algorithme :

- Initialisation

Soit X_o le point initial, nous prenons comme direction de montée, la direction du gradient Projeté :

$$D_1 = (\text{grad } f)_{X_o} - \sum_{i \in K_1} \lambda_i \, \varphi_i' \, (X_o).$$

On notera : $D = (\text{Grad P } f)_X$.

D_1 est calculée à l'aide du sous-programme Q GPROJ.
Nous cherchons le point X_1 sur la droite D_1 en calculant le pas h^*.

$$X_1 = X_o + h^* \cdot D_1$$

puis $$[\delta X_1] = X_1 - X_o = h^* D_1$$

et $$[\delta g_1] = (\text{gradPf})_{X_1} - (\text{GradPf})_{X_o}$$

On en déduit les trois matrices :

$$R_1 = \frac{[\delta X_1] \cdot {}^t[\delta X_1]}{{}^t[\delta g_1] \cdot [\delta X_1]}$$

$$S_1 = \frac{[\delta g_1] \cdot {}^t[\delta g_1]}{{}^t[\delta g_1] \cdot [\delta g_1]}$$

$$T_1 = I + R_1 - S_1.$$

On notera que $[\delta X_1]$, $[\delta g_1]$ sont des matrices colonnes, ${}^t[\delta X_1]$ et ${}^t[\delta g_1]$ sont des matrices lignes. Les numérateurs de R_1 et S_1 sont des matrices carrées symétriques $s \times s$, leurs dénominateurs sont des scalaires.

R_1, S_1 et T_1 sont donc des matrices carrées $s \times s$ symétriques.

- Itération d'indice r ($r \geq 2$)

Soit X_{r-1} le dernier itéré obtenu (X_1 à la suite de l'étape d'initialisation puis X_2, X_3, ...). Nous prenons comme direction de montée :

$$D_r = T_{r-1} \cdot (\text{Grad } f)_{X_{r-1}}.$$

Nous cherchons alors X, tableau ou point de la "droite de montée", défini par :

$$X = X_{r-1} + h \cdot D_r$$

qui rende f (X) maximum. La fonction $f(X_{r-1} + h D_r)$ est fonction de la seule variable h, fonction, que nous notons g (h), du second degré en h. Soit h^* la valeur qui rend g (h) maximum. On pourra obtenir h^* par la technique décrite en IV.3.c et mise en œuvre dans le sous-programme QSATUR.

Nous calculons alors :

$$X_r = X_{r-1} + h^* \cdot D_r$$

puis $\delta X_r = X_r - X_{r-1}$

et $\delta g_r = (\text{Grad P } f)_{X_r} - (\text{grad P } f)_{X_{r-1}}$.

On forme alors les matrices :

$$R_r = \frac{[\delta X_r] \cdot {}^t[\delta X_r]}{{}^t[\delta g_r] \cdot [\delta X_r]}$$

$$S_r = \frac{[T_{r-1} \cdot \delta g_r] \cdot {}^t[T_{r-1} \cdot \delta g_r]}{{}^t[\delta g_r] \cdot [T_{r-1} \cdot \delta g_r]}$$

puis $T_r = T_{r-1} + R_r - S_r$.

Comme dans le cas de l'initialisation, on notera que $[\delta X_r]$, $[\delta g_r]$, $[T_{r-1} \cdot \delta g_r]$ sont des matrices colonnes, leurs transposées sont donc des lignes. Ceci entraîne que R_r, S_r et T_r sont des matrices carrées symétriques.

- Sorties d'itérations :

Nous nous contentons de rappeler sans démonstration le résultat important suivant dans le cas des problèmes d'optimisation sans contraintes :

si $f(X) = f(o) + B \times X + \frac{1}{2} {}^tX \cdot A \cdot X$

le nième itéré T_n est identique à A^{-1} et, X_n est solution de :

$$\text{grad } f = B + A X = 0.$$

Dans notre cas avec contraintes linéaires, nous rappelons brièvement la caractérisation de l'optimum. A chaque étape nous testons si les conditions

de Kuhn et Tucker sont vérifiées, auquel cas si la direction de montée est nulle, l'optimum est obtenu.

Cette technique d'accélération de convergence est mise en œuvre dans le sous-programme DIRCON.

Remarque : sur les exemples traités par ordinateur, cette méthode a été trés efficace en réduisant le nombre d'itérations, par rapport à l'utilisation du programme sans accélération de convergence, de façon considérable.

CHAPITRE V

NOTICE D'UTILISATION DU PROGRAMME G R P Q (METHODES DU GRADIENT EN PROGRAMMATION QUADRATIQUE) :

1°) ENUMERATION DES POINTS COMMUNS ENTRE LES PROGRAMMES GRPL et GRPQ :

Nous ne rappellerons pas tous les points communs avec le programme GRPL de programmation linéaire, nous nous contenterons de les énumérer en indiquant la référence ;

- la gestion des variables et de l'état des contraintes est identique (cf. III.1) ainsi que les changements de base ;

- la liste des variables et tableaux utilisée dans le programme présentée dans le chapitre III. 2 sera complétée dans notre programme, cette liste étant commune aux deux programmes ;

- l'évolution de la matrice P^{-1} est identique dans les deux problèmes puisque la matrice P^{-1} intervient dans notre problème uniquement dans la phase "initialisation" pour la recherche d'une solution réalisable où la fonction objectif n'intervient pas. Par conséquent, tout ce qui concerne la programmation pour l'évolution de la matrice P^{-1} (QGPROJ et QSATUR) l'utilisateur devra se référer au cas linéaire (sous-programmes GRPROJ et SATUR).

- les sous-programmes INIT (Recherche d'une solution de base) et GRADUI (Calcul de la direction de montée par la méthode du Gradient Réduit "étendu") utilisés dans notre problème pour la phase "initialisation" sont ceux que l'on utilise dans le cas linéaire (cf. III 4 et 5) et que nous ne rappellerons pas ;

- les sous-programmes ARRAY et MINV d'inversion de matrice et MCADIR de résolution de systèmes linéaires sont des programmes de bibliothèque auxquels l'utilisateur peut se référer.

2°) LISTE DES VARIABLES ET TABLEAUX UTILISES :

a) la liste présentée ci-dessous est le complément de la liste du chapitre III. 2

Variables :

NOM	ROLE
CKT	Indicateur sur les conditions de Kuhn et Tucker CKT = { +1 : il y a au moins un coefficient de Kuhn et Tucker positif ; -1 : tous les coefficients sont négatifs ou nuls. Variable utilisée dans le sous-programme QGRPRØ et programme principal.
ISC	Indicateur si une nouvelle contrainte est saturée ISC = { 0 : le pas h est égal à h^*, il n'y a pas de nouvelle contrainte saturée ; 1 : il y a saturation d'une nouvelle contrainte. Variable initialisée dans QSATUR et intervenant pour l'utilisation du sous-programme DIRCON.

Tableaux :

NOM	DIMENSION	ROLE
CF	N × N	Contient les coefficients de la matrice définie positive de la fonction objectif.
TP	N	Contient les coefficients de la partie linéaire de la fonction objectif que l'on peut écrire sous forme matricielle : $F(X) = -\frac{1}{2} X \,.\, CF \,.\, X + TP \,.\, X$ Cette matrice CF et ce tableau TP ne sont pas nécessaires si l'on exprime la fonction objectif de façon explicite dans le sous-programme GRADF, ce qui permet un gain en place mémoire.
DO, DX, DG	NA	Tableaux de travail nécessaires au sous-programme DIRCØN.
T	NA × NA	

b) Données et paramètres d'entrée :

La lecture des données suivantes est faite au début du programme principal.

- Variables de dimensionnement : NA, N, NT, P.
- CF (N, N) et TP (N) définissant la fonction objectif.
- IVX (NT) : informations sur la nature des contraintes.
- V (NT, N) : coefficients des contraintes non obligatoirement mémorisables si à chaque utilisation de ces coefficients on les lit sur mémoire périphérique.
- X (N) : vecteur de départ est initialisé par l'utilisateur.

3°) PROGRAMME PRINCIPAL

a) Instructions de déclaration :

```
INTEGER    P, P1
CØMMØN /A/ P, N, NT, NA, M, INCI, IFI, P1, N1, D(N), ILU(NT), IVL(N)
           IVX(NT), EPS(2), C(N), IP(N), G(N, N), W(N, N), T1(N),
           T2(N), C1(N)
CØMMØN /B/ X(N), FX, H, V(NT, N), CF(N, N ), TP(N), T3(N), T4(N),
           CKT, ISC, ITER, INI.
```

b) Composition du programme :

Le programme comprend le programme principal et 9 sous-programmes

Deux sous-programmes ARRAY et MINV existant en bibliothèque permet tent d'inverser initialement la matrice W, ainsi que le sous-programme MCADIR qui permet de résoudre un système linéaire surdimensionné pour le calcul des coefficients de Kuhn et Tucker.

Cinq autres sous-programmes :

- GRADF : permet de calculer la valeur de la fonction objectif et de ses dérivées en un point X.
- INIT : permet la recherche d'une solution de base (cf. III.4).
- GRADUI : calcule la direction de montée par la méthode du Gradient Réduit "étendu" (cf. III.5).
- QGRPRØ : calcule la direction de montée par la méthode du Gradient Projeté, teste les conditions de Kuhn et Tucker et fait évoluer P^{-1} en cas de libération d'une contrainte.

- QSATUR : calcule le pas h et fait évoluer la matrice P^{-1} en cas de saturation d'une nouvelle contrainte.

- DIRCON : permet l'accélération de convergence dans la phase optimisation.

Remarque : Le programme peut être utilisé évidemment sans accélération de convergence.

c) Programme principal :

- Instructions de déclaration :

```
CØMMØN /A/ ...
CØMMØN /B/ ...
INTEGER P, P1.
```

- Lecture des données et initialisation

```
ITER = 0
EPS (1) = 1. E-6
EPS (2) = 1. E-4
N1 = N+1
P1 = P+1
```

- Phase "initialisation"

```
5       CALL INIT
        IF (M. LE. 0) GØ TØ 10
```

← Test si aucune contrainte n'est saturée, il n'y a pas d'inversion de matrice

```
        CALL ARRAY (2, M,M, NA, NA, W, W)
        CALL MINV (W, M, D1, T1, T2)
        CALL ARRAY (1, M, M, NA, NA, W, W)
```

} Inversion de la matrice W.

```
        IF (ABS(D1). GT. EPS(1)) GØ TØ 10
```

: Test sur le déterminant

```
        INCI = 10
        GØ TØ 40
10      IF (INI. EQ. 0) CALL GRADF (0)
```

Si le point X est réalisable calcul de $(Gradf)_X$

```
      CALL QGRPRØ
      IF (INCI.NE.0) GØ TØ 40
      IF (IFI.LE.0.AND.INI.EQ.1) GØ TØ 50
      IF (INI.EQ.1) GØ TØ 30
      DØ 20  I = 1,N                    Test si la direction de montée est nulle
      IF (ABS(D(I)).GT.EPS(2)) GØ TØ 30
20    CONTINUE
```

Test si l'optimum est obtenu :

```
      IF (CKT.LE.0.ØR.M.LE.0)  GØ TØ 25
      WRITE (6,620)
620   FORMAT (5X,'DIRECTION DE MONTEE NULLE : PAS DE SOLUTION')
      STOP 7
25    CALL GRADF (1)  ←Calcul de la valeur de la fonction objectif au
                       point X̄.
      WRITE (6,625) FX,  (I,X(I),I=1,N)
625   FORMAT (20X,'MAXIMUM OBTENU  FX =',E12.6,/(30X,1HX,I2,
     *3H = , E 15.7))
      STOP 2                    (fin normale du programme)
30    CALL QSATUR
      IF (INCI.NE.0) GØ TØ 40
      IF (INI.EQ.1) GØ TØ 5
```

Si la phase "initialisation"est terminée (INI = 0) on peut utiliser l'accélération de convergence en appelant le sous-programme DIRCØN

```
      CALL DIRCØN
      IF (INCI.NE.0) GØ TØ 40
      GØ TØ 10
40    WRITE (6,640)  INCI
640   FORMAT (5X,11HINCIDENT N0,I4)
      STOP 1
50    WRITE (6,650)
650   FORMAT (5X,26HPAS DE SOLUTION REALISABLE)
      STØP 2
      END
```

INCI = 10 : le déterminant de la matrice W est nul.

. Sous-programmes appelés : ARRAY, MINV, QGRPRØ, QSATUR, DIRCØN.

4°) SOUS-PROGRAMME QGRPRØ (ALGORITHME DU GRADIENT PROJETE)

Ce sous-programme calcule la direction de montée et teste les conditions de Kuhn et Tucker suivant que l'on se trouve dans la phase "initialisation" ou pas.

- Si INI = 1 : la phase "initialisation" n'est pas terminée.

  - il calcule les coefficients de Kuhn et Tucker et teste leur valeur : si un coefficient au moins est positif, il libère la contrainte associée à ce coefficient et fait évoluer la matrice P^{-1}. Dans tous les cas, il calcule la direction de montée par la méthode du Gradient Réduit "étendu" en appelant le sous-programme GRADUI.

- Si INI = 0 : la phase "initialisation" est terminée.

  - il teste les conditions de Kuhn et Tucker, deux cas peuvent se présenter :

  1er cas : les conditions sont vérifiées (tous les coefficients sont négatifs ou nuls, il calcule D par la méthode du Gradient Projeté

$$D = (\text{Grad } f)_X - \sum_{i \in K_1} \lambda_i \ \varphi_i' \ (X)$$

  2ème cas : les conditions ne sont pas vérifiées, on libère la contrainte associée au plus grand coefficient, on calcule de nouveau les coefficients $\bar{\lambda}_i$ de Kuhn et Tucker et enfin on détermine :

$$\vec{D} = (\text{Grad } f)_X - \sum_{i \in \bar{K}_1} \bar{\lambda}_i \ . \ \varphi_i' \ (X).$$

Pour le calcul des coefficients de Kuhn et Tucker, dans le cas INI = 0, on fait appel au sous-programme MCADIR qui permet de résoudre un système à m ($m \leq n$) inconnues λ_i correspondant aux m contraintes saturées avec n équations, système surdéterminé.

Paramètres d'entrée :

X : valeur du dernier itéré.

W : matrice inverse des coefficients des contraintes utiles correspondant aux variables de base.

INI : indication si X est réalisable ou non.

C : gradient de la fonction objectif au point X.

SUBROUTINE QGRPRØ

```
      CØMMØN /A/...
      CØMMØN /B/...
      INTEGER P, Pl
      IPASS = 0          Indicateur permettant de savoir dans la phase
                         "optimisation" (INI=0) si l'on se trouve dans le
                         2ème cas.
    5 CKT = 1
      INCI = 0
      IFI = 1
      IF (M.LE.0) GØ TØ 160   ← Aucune contrainte est saturée D = Gradf_X
      IF (INI.EQ.0) GØ TØ 30
Phase initialisation INI = 1
      IF (M.LT.N) GØ TØ 160
      DØ 10  J = 1, N
      Tl (J) = 0                 } Calcul des coefficients de Kuhn et
      DØ 10  I = 1, N              Tucker
   10 Tl(J) = Tl(J) + C(I) * W(I, J)
      GØ TØ 39
Phase optmisation (INI=0) - calcul des coefficients de Kuhn et Tucker :
   30 DØ 31  I = 1, N
```

```
31    T1(I) = C(I)
      DØ 33    J = 1, N
      J1 = J
      DØ 33    I = J1, N
      A = G (I, J)
      G (I, J) = G (J, I)
33    G (J, I) = A
      CALL MCADIR (G, T1, NA, N, M, T2, T3, T4, IP)
      IF (IP(1). LE. 0) GØ TØ 35
      INCI = 20
      RETURN
35    K = 0
      DØ 37  I = 1, NT
      IF (ILU(I). LE. 0) GØ TØ 37
      K = K+1
      DØ 36    J = 1, N
36    G (K, J) = V (I, J)
37    CONTINUE
      K1 = K+1
      DØ 38  I = K1, N
      DØ 38  J = 1, N
38    G (I, J) = 0
39    A = - 0.01
      IF (IPASS. EQ. 1)  GØ TØ 130
      K = P1
      DØ 40    I = K, M
      IF (T1(I). LE. A)  GØ TØ 40
      I1 = I
      A = T1(I)
40    CONTINUE
      IF (A. GT. 0) GØ TØ 45
      CKT = -1
```

Transposition de la matrice G

Reconstitution de la matrice G

(2ème cas de la phase optimisation)

Test des conditions de Kuhn et Tucker

(Tous les coefficients sont négatifs ou nuls)

```
      IF (INI.EQ.0) GØ TØ 130
45    K = 0
```

Libération de la contrainte correspondant au λ_j positif, choix de la variable sortante et évolution de la matrice W. Cette partie du programme est identique à celle de GRPRØJ (III.6), nous nous contentons de la rappeler.

```
      DØ 50       I = 1,NT
      IF (ILU (I).NE.1) GØ TØ 50
      K = K+1
      IF (K.EQ.I1) GØ TØ 60
50    CONTINUE
60    ILU(I) = 0
      IF (IVX(I).EQ.0)  GØ TØ 70
      J1 = IVX (I)
      GØ TØ 90
70    DØ 80       I = 1,N
      J1 = I
      IF (IVL(I).EQ.0) GØ TØ 80
      DØ 70       J=P1,NT
      IF (ILU(J).NE.1)  GØ TØ 75
      IF (IVX(J).NE.I) GØ TØ 75
      GØ TØ 80
75    CØNTINUE
      IF (ABS(W(J1,I1).GT.EPS(1)) GØ TØ 90
80    CONTINUE
      INCI = 30
      RETURN
90    IVL(J1) = 0
      M = M-1
      DØ 100   J = 1,N
      IF (J.EQ.I1) GØ TØ 100
      DØ 95     I = 1,N
      IF (I.EQ.J1) GØ TØ 95
```

```
      W(I, J) = W(I, J) - (W(J1, J) * W(I, I1)/W(J1, I1))
 85   CØNTINUE
100   CØNTINUE
      IK = 0
      DØ 110      I = 1, N
      IF (I. EQ. J1) GØ TØ 110
      IK = IK+1
      JK = 0
      DØ 105   J = 1, N
      IF (J. EQ. I1) GØ TØ 105
      JK = JK+1
      W (IK, JK) = W(I, J)
105   CØNTINUE
110   CØNTINUE
      K = 0
      DØ 120      I = 1, N
      IF (I. EQ. I1) GØ TØ 120
      K = K+1
      DØ 115      J = 1, N
115   G (K, J) = G (I, J)
120   CØNTINUE
      IF (INI. EQ. 1) GØ TØ 160
      IF (IPASS. EQ. 1) GØ TØ 130
      IPASS = 1
      GØ TØ 5
130   DØ 150      I = 1, N
      D (I) = 0
      DØ 140      K = 1, M
140   D (I) = D(I) - T1(K) * G(K, I)
150   D (I) = D(I) + C(I)
      RETURN
160   CALL GRADUI
      RETURN
      END
```

Si phase initialisation, on va calculer la direction Δ

Calcul de la direction de montée D par la méthode du gradient projeté

Calcul de la direction de montée Δ par la méthode du gradient réduit "étendu" dans la phase "initialisation"

. Sous-programme appelé GRADUI.

. INCI = Indicateur d'incident prend la valeur :

0 en cas de déroulement normal.

20 incident dans le sous-programme MCADIR.

30 impossiblité de libérer une variable, contraintes redondantes.

5°) SOUS-PROGRAMME GRADF : (Calcul du gradient de f (X))

Ce sous-programme calcule la valeur de la fonction objectif F et son gradient en un point donné. Les deux fonctions n'étant pas utiles à chaque appel de GRADF, il exécute suivant la valeur d'un paramètre INDIC l'une des deux fonctions.

Le paramètre INDIC passé en argument prend la valeur :

$$\text{INDIC} = \begin{cases} 1 & \text{GRADF : calcule la valeur de F au point X} \\ \text{Autres valeurs :} & \text{GRADF calcule le gradient de F en X.} \end{cases}$$

L'utilisateur a le choix entre donner l'expression mathématique de la fonction objectif F et de ses dérivées partielles ou de lire dans le programme principal la matrice CF (N, N) et le vecteur TP (N) qui définissent F.
$(F(X) = -\frac{1}{2}\ {}^{t}X \,.\, CF \,.\, X + {}^{t}TP \,.\, X)$.

Dans le second cas, le sous-programme GRADF peut se programmer de la façon suivante :

```
SUBRØUTINE GRADF (INDIC)
CØMMØN /A/...
CØMMØN /B/...
IF (INDIC.EQ.1) GØ TØ 30
DØ 20      I = 1,N
E = 0
```

} Calcul de $C = \overrightarrow{\text{Grad}}\ F(X)$

```
      DØ 10      J = 1, N
10    E = E - X (J) * CF (I, J)
20    C (I) = E+TP(I)
      RETURN
30    FX = 0
      DØ 40      I = 1, N
      T1 (I) = 0
      DØ 40      J = 1, N
40    T1 (I) = T1 (I) + (-0.5)  * X (J) * CF (I, J)
      DØ 50      I = 1, N
50    FX = FX + T1 (I) * X (I) + TP (I) * X (I)
      RETURN
      END.
```

6°) SOUS-PROGRAMME QSATUR (Calcul du pas h) :

Ce programme a pour but de calculer l'accroissement h. (cf. IV.3.c.).

. Si INI = 1, phase "initialisation" :

- il calcule le pas h en saturant une contrainte, il est identique au sous-programme SATUR (cf. III. 7).
  Il y a évolution de la matrice P^{-1} et gestion des variables de base.

. Si INI = 0, phase "optimisation" :

- le pas h est égal à h^* si $X_{r+1} = X_r + D_r h^*$ est un point du domaine réalisable.

$$h^* = \frac{dF}{dh} \quad (X_r + D_r h)$$

Paramètres :

. ISC = $\begin{cases} 1 & : 1 \text{ nouvelle contrainte est saturée. } (h = h_i) \\ 0 & : \text{il n'y a pas saturation de contrainte } (h = h^*) \end{cases}$

. ITER : compteur d'itérations.

. X : valeur du dernier itéré X_{r-1} en entrée et X_r en sortie

. D_{r-1} : direction de montée.

. W : matrice inverse des coefficients des contraintes "utiles" correspondant aux variables de base qui, si une contrainte est saturée, évolue par programme.

. INCI : Indicateur d'incident :

= 0 en cas de déroulement normal.

= 40 pas h infini (pas de solutions finies).

= 41 choix d'une variable entrante impossible (contraintes redondantes).

```
      SUBRØUTINE QSATUR
      CØMMØN /A/...
      CØMMØN /B/...
      INTEGER P, P1.
      MXITER = 50
      ITER = ITER + 1        (Incrémentation du nombre d'itérations)
      IF (ITER.GT.MXITER) STOP 6
      ISC = 0
      H1 = 1.E+30
      HETOIL = 0
```

Calcul du pas $h = h_i$ qui permet la saturation de la contrainte $\varphi_i(X)$.

```
      DØ 25      I = 1, NT
      IF (ILU(I).EQ.1) GØ TØ 25
      J = 1
      IF (ILU (I).GE. 0) GØ TØ 5
      J = -1
5     DØ 10   K = 1, N1
      IP (K) = 0
10    T1 (K) = J * T1 (K)
      E1 = 0
      E2 = 0
      DØ 15      J = 1, N
```

```
      E1 = E1 + T1 (J) * X (J)
15    E2 = E2 + T1 (J) * D (J)
      E1 = T1 (N1) - E1
      IF (E2.GE.0.ØR.E1.GE.0)  GØ TØ 25
      H = E1/E2
      IF (H.GT.H1) GØ TØ 25
      ISC = 1          ← Indicateur qu'une contrainte au moins est
                         saturée
      I1 = I
      DØ 20  K = 1, N1
20    T2 (K) = T1 (K)
      H1 = H
25    CONTINUE
      H = H1
      IF (INI.EQ.1) GØ TØ 40     (Si phase, "initialisation" le calcul de h*
                                  est inutile).
Calcul de h*
      A = 0.
      DØ 35      I = 1, N
      T1 (I) = 0
      DØ 30      J = 1, N
30    T1 (I) = T1 (I) + D (J) * CF (I, J)
      HETOIL = HETOIL - T1 (I) * X (I) + TP (I) * D (I)
35    A = A + T1 (I) * D (I)
      HETOIL = HETOIL/A
      IF (HETOIL.LE.0.ØR.HETOIL.GE.H) GØ TØ 40
      ISC = 0  ← Indicateur que h = h*
      H = HETOIL
      GØ TØ 50
40    IF (ISC.GE.1)  GØ TØ 45
      INCI = 40        ← Incident h n'est pas égal à h* et aucune contrain-
                         te n'a pu être saturée.
      RETURN
45    ILU (I1) = 1
      M = M+1          (Incrémentation du nombre de contraintes saturées).
```

```
      Calcul du nouvel itéré X_r
   50 FX = 0
      DØ 60      I = 1,N
      X (I) = X (I) + D (I) * H
   60 FX = FX + X (I) * C (I)   ← Calcul de FX dans le cas de la phase
      IF (INI.EQ.1) GØ TØ 70          "initialisation"
      CALL GRADF (1)          ← Calcul de FX dans le cas de la phase
                                      "optimisation"
   70 WRITE (6,670) ITER, FX
  670 FØRMAT (37X, I2, 8X, E12.7)
      IF (INI.EQ.1.ØR.HETOIL.GT.0) RETURN
      Toute la suite des instructions du programme nécessaire à la gestion
      des variables de base et l'évolution de la matrice W est identique au
      programme SATUR (cf. III.7).
      .........................
      END
```

Sous-programme appelé GRADF.

7°) SOUS-PROGRAMME DIRCØN (Accélération de convergence)

Ce programme met en œuvre la méthode des directions conjuguées (cf. IV.5) qui permet l'accélération de convergence dans la phase "optimisation".

Paramètres :

D_{r-1} : direction de montée calculée à la dernière itération.

ISC : indicateur si une contrainte a été saturée lors de la dernière itération

- = 0 il n'y a pas eu saturation ($h = h^*$)
- = 1 il y a eu saturation.

IS = N-M : nombre maximum d'étapes dans le cycle.

ITER1 : compteur d'itération ou d'étapes dans le cycle.

EPS (1) : valeur ε qui permet de tester si D = 0.

```
      SUBRØUTINE DIRCØN
      CØMMØN /A/...
      CØMMØN /B/...  (remplacer T4 par TDG)
      INTEGER P, P1
      DIMENSION DO(NA), DX(NA), DG(NA), T(NA, NA)
      IF (ISC.NE.0) RETURN    (S'il y a eu saturation de contrainte,
      IF (M.GE.(N-1)) RETURN   l'utilisation de DIRCØN est inutile)
```

Etape préliminaire

```
      DØ 5      I = 1, N
5     DO (I) = D (I)
      CALL GRADF (0)  ← Calcul du Gradient de F en X_r.
      CALL QGRPRØ     ← Calcul de la direction de montée à partir de X_r
                        et utilisant la technique du gradient projeté.
      J = 0
      DØ 10     I = 1, N
      IF (IVL (I).NE.0) GØ TØ 10
      J = J+1
      DX (J) = H * DO (I)
      DG (J) = D (I) - DO(I)
10    DO (I) = D (I)
      IS = N-M        ← (IS = nombre de variables libres)
      DENR = 0
      DENS = 0
      DØ 20     J = 1, IS
      DENR = DENR + DX (J) * DG (J)
20    DENS = DENS + DG (J) * DG (J)
      DØ 35     J = 1, IS
      DØ 35     K = 1, IS
      IF (J.EQ.K) GØ TØ 25
```

Calcul de T = I + R - S

```
      T (J,K) = 0
      GØ TØ 30
25    T (J,K) = 1
30    T (J,K) = T(J,K) + DX(J) * DX (K)/DENR - DG(J)*DG(K)/DENS
35    CØNTINUE

      ITER1 = 1
```

Cycle d'itérations :

```
40    ITER1 = ITER1 + 1
      II = 0
      DØ 50       I = 1,N
      IF (IVL(I).NE. 0) GØ TØ 45
      S = 0
      KK = 0
      II = II + 1
      DØ 42       K = 1,N
      IF (IVL(K).NE. 0) GØ TØ 42
      KK = KK+1
      S = S + T (II,KK) * DØ (K)
42    CØNTINUE
      T3 (I) = S
      D (I) = S
      T1 (II) = S
      GØ TØ 50
45    CØNTINUE
50    CØNTINUE
```

Elimination $D_B = f(D_H)$ pour compatibilité de D avec les contraintes saturées.

Si aucune contrainte saturée $D = D_H$

```
      IF (M. LE. 0) GØ TØ 80
      DØ 60       I = 1,M
```

```
      T2 (I) = 0
      K = 0
      DØ 55       J = 1, N
      IF (IVL (J). EQ. 1) GØ TØ 55
      K = K+1
      T2 (I) = T2 (I) + G (I, J) * T1 (K)
55    CØNTINUE
60    CØNTINUE
      DØ 65       I = 1, M
      T1 (I) = 0
      DØ 65       J = 1, M
65    T1 (I) = T1 (I) + W (I, J) * T2 (J)
      K = 0
      DØ 70       I = 1, N
      IF (IVL (I). EQ. 0)  GØ TØ 70
      K = K+1
      D (I) = -T1 (K)
70    CØNTINUE
80    CØNTINUE
```

Détermination d'un nouveau point $X_{r+1} = X_r + D_r * h$.

```
      CALL QSATUR
      IF (INCI. NE. 0. ØR. ISC. EQ. 1)  RETURN
      IF (ITER1. EQ. IS) RETURN       (Test de fin de cycle)
      CALL GRADF (0)
      CALL QGRPRØ
      DØ 85       I = 1, N                    }
      IF (D(I). GT. EPS(1))  GØ TØ 90         }  Test si D = 0
85    CØNTINUE                                }
      RETURN
90    J = 0
      DØ 95       I = 1, N
      IF (IVL (I). NE. 0)  GØ TØ 95
```

```
      J = J+1
      DX (J) = H * T3 (I)
      DG (J) = D (I) - DO  (I)
 95   DO  (I) = D (I)
      DØ 105  I = 1, IS
      S = 0
      DØ 100      K = 1, IS
100   S = S + T(I, K) * DG (K)
105   TDG (I) = S
      DENR = 0
      DENS = 0
      DØ 110      I=1, IS
      DENR = DENR + DX (I) * DG (I)
110   DENS = DENS + DG (I) * TDG (I)
      DØ 115      I = 1, IS
      DØ 115      K = 1, IS
115   T(I, K) = T(I, K) + DX(I) * DX (K)/DENR-TDG(I)*TDG(K)/DENS
      GØ TØ 40
      END
```

Sous-programmes appelés GRADF, QGRPRØ, QSATUR.

8°) <u>EXEMPLES TRAITES PAR PROGRAMME</u> :

Parmi les exemples qui nous ont permis de tester le programme, nous en présentons deux, ci-dessous, où la position de l'optimum par rapport au domaine réalisable est différente.

Dans le premier exemple, l'optimum est à l'intérieur du domaine défini par les contraintes ; dans le second exemple, il est sur la frontière du domaine.

Chaque problème a été traité avec et sans accélération de convergence.

a) 1er exemple :

Soit à maximiser la fonction :

$$f(X) = -123.08\, x_1^2 - 203.64\, x_2^2 - 182.25\, x_1 x_2 + 138.08\, x_1 + 282.92$$

sous les contraintes suivantes :

$$\begin{cases} x_1 \geq 1 \\ -x_1 \geq -2.5 \\ x_2 \geq 1 \\ -x_2 \geq -2. \end{cases}$$

Rappelons que l'optimum est atteint lorsque les conditions de Kuhn et Tucker sont vérifiées et la norme du vecteur D est inférieure à un ε donné. ε est choisi égal à 10^{-4}.

- Exécution du programme sans accélération de convergence (s.p. DIRCON)

Un point réalisable évident est l'origine : $X_0 = (0, 0)$.

Itération 1 :

INI = 0 : indicateur que X_0 est réalisable.

$X_1 = (0.5867,\ 0.1826)$ $\qquad F(X_1) = 54.8632.$

Le tableau ILU concernant l'information sur l'état des contraintes indique aucune contrainte saturée.

Itération 2 :

$X_2 = (0.3555,\ 0.4836)$ $\qquad F(X_2) = 67.2114.$

Itération 3 :

$X_3 = (0.2775\ ,\ 0.4237)$ $\qquad F(X_3) = 69.5415.$

Itération 4 :

$$X_4 = (0.2339 , 0.4805) \qquad F(X_4) = 69.9812.$$

Itération 5 :

$$X_5 = (0.2192 , 0.4692) \qquad F(X_5) = 70.0642.$$

Ensuite la valeur de la fonction au point X_i évolue faiblement jusqu'à l'obtention de l'optimum à l'itération numéro 17.

Itération 17 :

$$X_{17} = (0,2056 , 0.4798) \qquad F(X_{17}) = 70.083.$$

L'indicateur ILU indique toujours aucune contrainte saturée. Par conséquent, l'optimum est à l'intérieur du domaine réalisable.

Le fait que la valeur de la fonction évolue faiblement entre les itérations 5 et 17 nous a amené à constater que la méthode du gradient ne converge pas rapidement. Pour accélérer la convergence des itérations, on a utilisé la méthode des directions conjuguées :

- Exécution du programme avec accélération de convergence :

L'origine étant toujours le point de départ.

Itération 1 :

INI = 0

$$X_1 = (0.5867 , 0.1826) \qquad F(X_1) = 54.8632.$$

L'itération 1 est identique à l'exécution précédente. Mais l'utilisation du sous-programme DIRCON ensuite nous donne immédiatement l'optimum à la seconde itération.

$$X_2 = (0.2056 , 0.4798) \qquad F(X_2) = 70.083.$$

L'accélération de convergence évite l'oscillation autour de l'optimum et elle est donc très efficace. Il y a un gain de 15 itérations.

b) 2ème exemple :

Soit à maximiser la fonction :

$$F(X) = -\frac{1}{2}(2x_1^2 + 10x_2^2 + 12x_3^2 + x_4^2) + 2x_1x_2 + 2x_1x_3 - 6x_2x_3 + 6x_1 + 2x_2 + x_4$$

sous les contraintes suivantes :

$$\begin{cases} -13.5x_1 + 15.5x_2 - 16.x_3 + x_4 = -30 \\ -16.x_1 - 8.0x_2 - 4.x_3 + x_4 \geq -75 \\ -x_1 - x_2 - x_3 + x_4 \geq -6 \\ x_1 \geq 0 \quad x_2 \geq 0 \quad x_3 \geq 0 \quad x_4 \geq 0. \end{cases}$$

Le point de départ : $X_0 = (-20, -30, 0, -10)$
est volontairement choisi quelconque pour tester la partie du programme (INIT) qui recherche une solution réalisable.

- Exécution sans accélération de convergence :

On note : FA fonction objectif pendant la phase de recherche d'une solution réalisable.

Itération 1 :

INI = 1 : indicateur signalant que X_0 n'est pas réalisable.

$X_1 = (-25.3656 , -22.6224 , 0 , -9.1057)$

$FA(X_1) = -275.$

ILU = 1, 0, 0, -1, -1, 1, -1.

Le tableau ILU indique que la première contrainte est vérifiée donc saturée puisque bilatérale. La 6ème contrainte est saturée et la 4ème, 5ème et 7ème ne sont pas vérifiées. Les 2ème et 3ème sont vérifiées.

Itération 2 :

INI = 1

$X_2 = (-2.5636,\ -4.0856,\ 0,\ -0.119.10^{-6})$ $\quad FA(X_2) = -57.0936$

ILU = 1, 0, 0, -1, -1, 1, 1

Une contrainte de plus a été saturée : la 7ème.

Itération 3 :

INI = 1

$X_3 = (-0.298.10^{-7},\ -1.9354,\ 0,\ -0.1192.10^{-6})$ $\quad FA(X_3) = -6.6493$

ILU = 1, 0, 0, 1, -1, 1, 1.

La contrainte numéro 4 est saturée.

Itération 4 :

INI = 1

$X_4 = (-0.298.10^{-7},\ -0.149.10^{-7},\ 1.8181,\ -0.119.10^{-6})$

$FA(X_4) = -1.9354$

ILU = 1, 0, 0, 1, 1, 0, 1.

La contrainte numéro 5 est saturée, et la 6ème est libérée. Ainsi la phase "initialisation" est terminée.

Itération 5 :

INI = 0 $\quad X_4$ est vecteur réalisable.

$X_5 = (2.1289,\ -0.762.10^{-7},\ 0.14079,\ -0.144.10^{-6})$

$F(X_5) = 8.7219.$

Itération 6 :

$X_6 = (2.3219,\ 0.4226,\ 0.3857,\ -0.144.10^{-6})$ $\quad F(X_6) = 10.3755.$

Itération 7 :

$X7 = (2.7413,\ 0.4195,\ 0.0606,\ 0.1351)$ $\quad F(X_7) = 11.476.$

Itération 14 :

$X14 = (3.4424,\ 0.96032,\ 0.04628,\ 0.6307)$

$F(X_{14}) = 13.19649.$

Itération 37 :

$X_{37} = (3.6016,\ 1.0190,\ 0.0025,\ 1.0606)$ $\quad F(X_{37}) = 13.32474.$

A partir de cette étape, la valeur de la fonction n'évolue pratiquement plus. Le vecteur de montée vaut :

$$D = (-0.16458.10^{-4},\ 0.78964.10^{-3},\ 0.97142.10^{-3},\ 0.35715.10^{-2}).$$

Si l'on se contente de la précision 10^{-2} on peut arrêter le déroulement du programme à cette itération.

Pour une précision de 10^{-3}, il faut itérer jusqu'à la 47ème étape.

Pour une précision de 10^{-4}, il faut itérer jusqu'à la 113ème étape.

On obtient à l'itération 113 :

$$X_{113} = (3.6019,\ 1.0190,\ 0.00209,\ 1.0644)$$

$$F(X_{113}) = 13.32475.$$

Remarque : La 1ère contrainte qui est bilatérale reste vérifiée pendant toutes ces itérations et l'optimum se trouve donc sur la frontière du domaine réalisable.

- Exécution avec accélération de convergence :

La phase "initialisation" est identique puisque la fonction objectif du problème n'intervient pas et il n'y a pas accélération de convergence pendant cette phase.

Les cinq premières itérations sont identiques.

Itération 6 :

$$X_6 = (2.321,\ 0.422,\ 0.385,\ -0.14.10^{-6}) \qquad F(X_6) = 10.3755$$

$$ILU = (1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 1).$$

Le nombre de variables libres est égal à 2 (N-M).

Cette itération est l'étape préliminaire avant d'exécuter le cycle de s (s = 2) itérations où intervient l'accélération de convergence.

Itération 7 :

$$X_7 = (2.951,\ 0.528,\ 0.148.10^{-4},\ 0.186) \qquad F(X_7) = 11.9503.$$

Au cours de cette itération, il y a saturation de la contrainte 6, le cycle est interrompu pour passer à une autre étape préliminaire.

Itération 8 :

$$X_8 = (3.009,\ 0.744,\ 0.161,\ 0.246) \qquad F(X_8) = 12.508$$

$$ILU = (1,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0).$$

La contrainte 6 est libérée ; le nombre de variables libres est égal à 3, nombre d'itérations du prochain cycle.

Itération 9 :

$$X_9 = (3.450,\ 0.924,\ -0.8.10^{-6},\ 0.514) \qquad F(X_9) = 13,136.$$

Il y a de nouveau saturation de la contrainte 6 et par conséquent interruption du cycle.

Itération 10 :

$$X_{10} = (3.471,\ 0.985,\ 0.043,\ 0.577) \qquad F(X_{10}) = 13,179.$$

Il y a libération de la contrainte 6.

$S = 3.$

Itération 11 :

$$X_{11} = (3.586,\ 1.021,\ 0.7.10^{-6},\ 0.785) \qquad F(X_{11}) = 13,285.$$

De nouveau, saturation de la contrainte 6.

Itération 12 :

$$X_{12} = (3.591,\ 1.008,\ 0.2.10^{-7},\ 1.059) \qquad F(X_{12}) = 13,324.$$

A partir de cette itération, la valeur de la fonction objectif n'évolue plus et on a $\|D\| \leq 10^{-1}$.

Pour obtenir une meilleure précision sur le vecteur X et pour que la norme du vecteur D soit inférieure à 10^{-2}, il faut aller jusqu'à l'itération 15.

Pour $\|D\| \leq 10^{-3}$, il faut itérer jusqu'à la 21ème étape.

Pour $\|D\| \leq 10^{-4}$, il faut itérer jusqu'à la 25ème étape.

Remarques :

Dans cet exemple, l'algorithme d'accélération de convergence est moins efficace que dans l'exemple précédent car il est interrompu par la saturation de la contrainte numéro 6 qui est définitivement libérée à partir de l'itération numéro 13.

Il y a tout de même un gain de 25 itérations pour l'obtention de l'optimum par rapport à l'exécution de convergence.

Notons que des erreurs de chute de précision dans les calculs entravent parfois la rapidité de convergence des algorithmes.

BIBLIOGRAPHIE

DESBAZEILLE G. Exercices et problèmes de recherche opérationnelle. Dunod, Paris, 1964.

FAURE R. Eléments de la recherche opérationnelle. Gauthier-Villars, Paris.

KAUFMANN A. Méthodes et modèles de la recherche opérationnelle. Dunod, Paris, 1964.

KÜNZI H.P. - KRELLE W. et OETTLI W. La programmation non linéaire. Gauthier-Villars, Paris.

LEGRAS J. Algorithmes d'optimisation en problèmes avec contraintes et en contrôle optimal. Université de Nancy I, cours de D.E.A., 197[illegible]

SIMONNARD M. Programmation linéaire. Dunod, Paris, 1962.

NOM DE L'ETUDIANT : ATLAN Daniel

NATURE DE LA THESE : DOCTORAT DE SPECIALITE en INFORMATIQUE

VU, APPROUVE

et PERMIS D'IMPRIMER

NANCY LE 21 novembre 1977

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE DE NANCY

Le Président

M. BOULANGE