TER 2008-2009 MASTER 1 INFORMATIQUE
Option CASAR

Visualisation en 2D et 3D de la succession des étapes issues de la résolution d'un programme linéaire par la méthode du simplexe

Encadrant :

- **Mr Rodolphe GIROUDEAU**

Membres du groupe :

- **BENYAHIA Zakaria**
- **GALINDO Benjamin**
- **LAGHA Nazim**

Abstract

Ce document est le fruit du travail accompli dans le cadre des **T**ravaux d'**É**tudes et de **R**echerche, requis dans notre formation de MASTER 1ère année INFORMATIQUE option **C**ombinatoire **A**lgorithmique **S**ystème **A**dministration **R**éseaux. Il reflète par ailleurs les connaissances acquises durant nos années de formation antérieures, notamment en MATHEMATIQUES.

Nos travaux consistent en la mise en œuvre d'une interface graphique deux dimensions, ainsi qu'une plateforme trois dimensions, représentant respectivement dans $\Re^2$ et $\Re^3$ les étapes intermédiaires de la résolution d'un problème de programmation linéaire.

La résolution du problème linéaire en amont de la représentation graphique, se base sur l'algorithme du simplexe.

L'objectif de cette étude est de constater le déplacement dans l'enveloppe convexe, des solutions intermédiaires de l'algorithme.

La concrétisation de cette étude a eu lieu grâce à l'étroite collaboration de notre encadrant **Mr Rodolphe Giroudeau**. Nous tenons donc à le remercier de nous avoir aiguillés et soutenus constamment durant sa réalisation.

Par ailleurs, on remercie toute personne, qui par un conseil, une information, ou un effort de disponibilité, a contribué de près ou de loin au bon déroulement de nos travaux.

" C'est la manière dont nous faisons ce que nous faisons qui confère un sens à ce qui est fait."
Richard SERRA, Écrits et entretiens 1970-1989. Daniel Lelong éditeur.

SOMMAIRE

1 Introduction

Ce document fait l'objet d'un rapport réalisé dans le cadre des **T**ravaux d'**É**tudes et de **R**echerche requis dans notre formation de MASTER 1ère année. Ils ont pour but d'une part de nous introduire dans une démarche autonome de recherche d'informations, et d'autre part de nous initier à la maitrise des premiers éléments de gestion et de réalisation de projets.

Les modalités de recherche sont fixées par l'enseignant encadreur, elles définissent le cadre précis du travail à effectuer, à savoir : thème, objectif, et volume horaire. Il nous est en revanche demandé de définir : bibliographie, méthodes, formats, et supports.

Le déroulement des travaux comprend deux étapes essentielles. La réalisation du Cahier Des Charges constitue la première, durant celle-ci nous avons cerné le sujet proposé, les outils nécessaires à son élaboration, ainsi que la méthodologie de travail envisagée et la répartition temporelle des taches, sur les éléments constituant le groupe de travail. La deuxième comprend la mise en œuvre des travaux, et donc la concrétisation en dernière instance du projet. Les descriptions suivantes concernent principalement le déroulement de la deuxième phase.

1.1 Présentation du sujet

Nos travaux portent sur la réalisation d'interfaces graphiques, représentant les solutions intermédiaires de la résolution d'un problème de programmation linéaire, par la méthode du Simplexe. Les problèmes envisagés sont dans $\Re^2$ ou dans $\Re^3$, en d'autres termes, les illustrations graphiques seront respectivement en deux ou trois dimensions.

1.2 Objectifs à atteindre

L'objectif de cette étude est de constater le déplacement dans l'enveloppe convexe, des solutions intermédiaires de l'algorithme du simplexe.

L'étude théorique du sujet faite durant l'élaboration du cahier des charges, nous a amenés à considérer que la réalisation globale du projet, passée par la concrétisation des trois principaux objectifs intermédiaires suivants :

- L'implémentation de l'algorithme du simplexe
- La réalisation d'une interface graphique consacrée aux problèmes dans $\Re^2$
- La réalisation d'une plateforme graphique dédiée aux problèmes dans $\Re^3$

2 La programmation linéaire

La programmation linéaire constitue un problème fondamental en optimisation mathématique ; il s'agit de déterminer l'optimum (maximum ou minimum) d'une fonction linéaire tout en respectant des contraintes.

Beaucoup de problèmes réels de recherche opérationnelle peuvent être exprimés comme un problème de programmation linéaire. C'est pourquoi un grand nombre d'algorithmes pour la résolution d'autres problèmes d'optimisation sont fondés sur la résolution de problèmes linéaires.

Domaines d'application:

Les graphes obtenus à partir des contraintes imposées fournissent une modélisation « naturelle » des réseaux. Certains algorithmes qui leurs sont appliqués permettent alors de résoudre de nombreux problèmes de base, comme la détermination de routage optimaux en Informatique ou l'attribution optimale de tâches à des employés en fonction de leurs aptitudes ou de leurs désirs.

La programmation linéaire est donc devenue un concept essentiel. En effet, son application s'implique désormais dans le fonctionnement de la majorité des entreprises, d'où l'importance cruciale qu'elle détient dans divers environnements de production.

3 L'algorithme du simplexe

3.1 Présentation

L'algorithme du simplexe conçu par George Danzig en 1947 est une technique fondamentale, pour la résolution des problèmes de programmation linéaire. Ainsi, étant donné un ensemble d'inégalités linéaires sur *n* variables réelles, l'algorithme permet de trouver la solution optimale pour une fonction objectif, qui est elle aussi linéaire.

D'autres méthodes de résolution du même type de problème ont été conçues, parmi elles l'algorithme de L.G. Khachiyan (1979) et celui de N.Karmarkar (1984). Ces deux derniers abordent, en particulier, divers problèmes rencontrés lors de l'application de la méthode du simplexe, à savoir le risque de cyclage entrainé par la dégénérescence.

Néanmoins, l'algorithme du simplexe est très efficace en pratique et est implémenté dans tous les solveurs de programmes linéaires.

3.2 Principe

Les contraintes du problème linéaire sont représentées par des inégalités linéaires. L'ensemble de ces dernières forme d'un point de vue géométrique un polytope convexe dans l'espace à *n* dimensions, et plus précisément un polygone en deux dimensions et un polyèdre en trois dimensions.

Dans l'algorithme du simplexe, on ne considère que des solutions de base réalisables, ce qui revient géométriquement à ne prendre en compte que les sommets du polytope des contraintes. Ainsi, lors de la résolution, il s'agira de trouver le sommet optimal pour la fonction de coût donnée.

L'idée de l'algorithme consiste à démarrer d'un sommet quelconque du polytope et, à chaque itération, d'aller à un sommet adjacent s'il est possible d'en trouver un meilleur pour la fonction objectif. S'il n'y en a pas, l'algorithme s'arrête en concluant que le sommet courant est optimal.

En général, il y a plusieurs sommets adjacents au sommet courant qui sont meilleurs pour l'objectif. Il faut en sélectionner un seul, la règle de sélection est appelée règle de pivotage.

3.3 Cas défavorables

Il y a deux cas de programmation linéaire où il n'existe pas de solution optimale, à savoir :

Le premier est lorsque les contraintes se contredisent mutuellement, par exemple : ((x ≥ 2) et (x ≤ 1)). Dans un tel cas, le polytope est vide et il n'y a pas de solution optimale, puisqu'il n'y a pas de solution du tout. Le programme linéaire est alors dit *infaisable.*

Le deuxième concerne les polyèdres non-bornés dans la direction définie par la fonction objectif, par exemple : $\mathbf{z = x_1 + x_2}$ tel que $\mathbf{x_1 \geq 0}$, $\mathbf{x_2 \geq 0}$ et $\mathbf{x_1 + x_2 \geq 10}$. Dans ce cas, il n'y a pas de solution optimale puisqu'il est possible de construire des solutions satisfaisant les contraintes avec des valeurs arbitrairement élevées de la fonction objectif. En somme, ce genre de problèmes admettent des solutions réalisables, mais n'ont pas de valeur optimale, ils sont donc dits *non-bornés*.

En dehors de ces deux cas rarissimes dans les problèmes pratiques, l'optimum est toujours atteint à un sommet du polytope. Cependant, il n'est pas nécessairement unique. En effet, il est possible d'avoir un ensemble de solutions optimales correspondant à une arête ou à une face du polytope, voire au polytope en entier.

3.4 Convexité

Le terme convexe est justifié par la définition suivante :

Soient $M = (x_{1, \ldots,} x_n)$ et $P = (y_1,\ldots, y_n)$ deux points quelconques du polytope (ou du polyèdre) déterminé par les contraintes; alors, quelque soit le réel λ avec $0 \leq \lambda \leq 1$, le point $\lambda M + (1 - \lambda) P$ *(de coordonnées* $\lambda x_i + (1 - \lambda) y_i$*)* appartient au polytope (ou au polyèdre). Les n-uplets $(x_{1, \ldots,} x_n)$ qui satisfont les contraintes s'appellent *solutions réalisables* du problème. Ce sont les coordonnées des points intérieurs au polyèdre ou au polytope des contraintes.

Le théorème qui suit corrobore les explications précédentes :

THEOREME. *On considère une forme linéaire des n variables* $x_{1, \ldots,} x_n$ *soumises à des contraintes linéaires. Son maximum, qui existe si cette forme est majorée, est atteint au moins en un sommet du polyèdre des contraintes.*

3.5 Forme standard

Dans nos travaux, nous n'avons considéré que des problèmes de programmation linéaires proposés sous leur forme standard. Dans ce qui suit, nous allons définir la forme en question.

Soit un problème d'optimisation linéaire à ***n*** variables et ***n+m*** contraintes qui s'écrit comme ci-dessous :

$$\textbf{\textit{Maximiser}}\ z = \sum_{j=1}^{n+m} c_j x_j$$

$$\textbf{avec les contraintes :}\ \begin{cases} \sum_{j=1}^{n+m} a_{ij} x_j = b_i & pour : \quad i = 1,\ldots,m \\ x_j \geq 0 & pour : \quad j = 1,\ldots,n+m \end{cases}$$

3.5.1 Notation

On pose :

- $A = \left(a_{ij}\right)_{i=1\ldots m,\, j=1\ldots n+m}$
- $X = (x_1,\ldots,x_{n+m})^t$
- $c = (c_1, c_2,\ldots,c_{n+m})$
- $b = (b_1,\ldots,b_m)^t$

Notre problème s'écrit maintenant :

$$\textbf{Maximiser}\ c.X\ \textbf{avec}\ A.X = b\ \textbf{et}\ X \geq 0\,\textbf{.}$$

Remarquons que cette définition généralise celle d'un problème de programmation linéaire mis sous forme standard, après introduction des variables d'écart.

Suite des notations

$x_{j_1},...,x_{j_n}$ étant n des $n+m$ variables et $x_{j_{n+1}},...,x_{j_{n+m}}$ les autres variables, on pose :

- $X_B = (x_{j_{n+1}},...,x_{j_{n+m}})^t$
- $X_N = (x_{j_1},...,x_{j_n})^t$
- $B =$ La matrice formée par les m colonnes de A correspondant aux coefficients de $x_{j_{n+1}},...,x_{j_{n+m}}$ (en respectant l'ordre des variables)
- $A_N =$ La matrice formée par les n colonnes de A correspondant aux coefficients de $x_{j_1},...,x_{j_n}$ (en respectant l'ordre des variables).
- $c_B =$ La matrice-ligne formée des coefficients dans z de $x_{j_{n+1}},...,x_{j_{n+m}}$ (en respectant l'ordre des variables).
- $c_N =$ La matrice-ligne formée des coefficients dans z de $x_{j_1},...,x_{j_n}$ (en respectant l'ordre des variables).

3.5.2 Base

Définition. Une base est constituée de m variables qui s'expriment de façon unique, et affine, en fonction des n autres variables, cette expression étant algébriquement équivalent aux m contraintes d'égalité initiales.

Propriété. $x_{j_1},...,x_{j_m}$ est une base si et seulement si la matrice B correspondante est inversible.

3.5.3 Variable en base et variable hors-base

Tout n-uplet de valeurs $(x_1^*,...,x_n^*)$ satisfaisant les contraintes constitue une *solution réalisable.*

La fonction z est dite *fonction objectif*. Les n variables $(x_1,...,x_n)$ sont appelées *variables de décision, variable de choix,* ou encore *variable hors-base*; les m variables $(x_{n+1},...,x_{n+m})$ quant à elles s'appellent *variables d'écart* ou bien *variables de base*. Une fonction réalisable qui maximise la fonction objectif est dite *solution optimale*.

3.5.4 Initialisation de l'algorithme du simplexe

Une base étant choisie, la solution basique associée est obtenue en donnant à toutes les *variables hors-base* la valeur 0, c'est-à-dire par $X_N^* = 0$. Les valeurs prises par les *variables de base* sont alors données par:

$$X_B^* = B^{-1}b$$

La base est dite réalisable si toutes les coordonnées de X_B^* sont positives ou nulles. Trouver une base réalisable revient à déterminer une matrice carrée B de dimension (n, n), extraite de A, inversible et telle que $B^{-1}b$ ait tous ses coefficients positifs ou nuls. Une telle base nous permettra d'initialiser l'algorithme du simplexe.

On peut remarquer que, lorsque le problème initial est donné sous forme standard, la matrice carrée des coefficients des variables d'écart est l'identité; c'est donc une matrice inversible: **on peut prendre comme base l'ensemble des variables d'écart. Cette base est réalisable si le vecteur b n'a aucune composante négative**.

3.6 Algorithme

Nous avons choisi d'implémenter la méthode matricielle de l'algorithme du simplexe. Dans ce qui suit, on présente la résolution d'un problème de programmation linéaire, basée sur cette méthode.

Considérons le problème :

Maximiser c. *X* sous les contraintes *A.X* = *b* et *X* ≥ 0

Supposons connue une base réalisable. On appelle c_B et *B* les parties de *c* et de *A* correspondant à cette base réalisable X_B. Il nous faut initialiser X_B^* qui vaut $B^{-1}b$.

Une étape de l'algorithme du simplexe se déroule alors comme suit :

1. On résout : $y.B = c_B$ avec $y = \begin{pmatrix} y_1 & y_2 \end{pmatrix} \in \Re^2$

2. Calcul de la variable entrante: Choisir une colonne entrante s'il en existe une, c'est-à-dire une colonne a_j de *A* non dans *B*, telle que $y.a_j$ soit plus petit que c_j (s'il s'agit de minimiser, on chercherait une colonne a_j telle que $y.a_j$ soit plus grand que c_j). S'il n'existe pas de colonne entrante, la base est optimale et l'algorithme s'arrête.

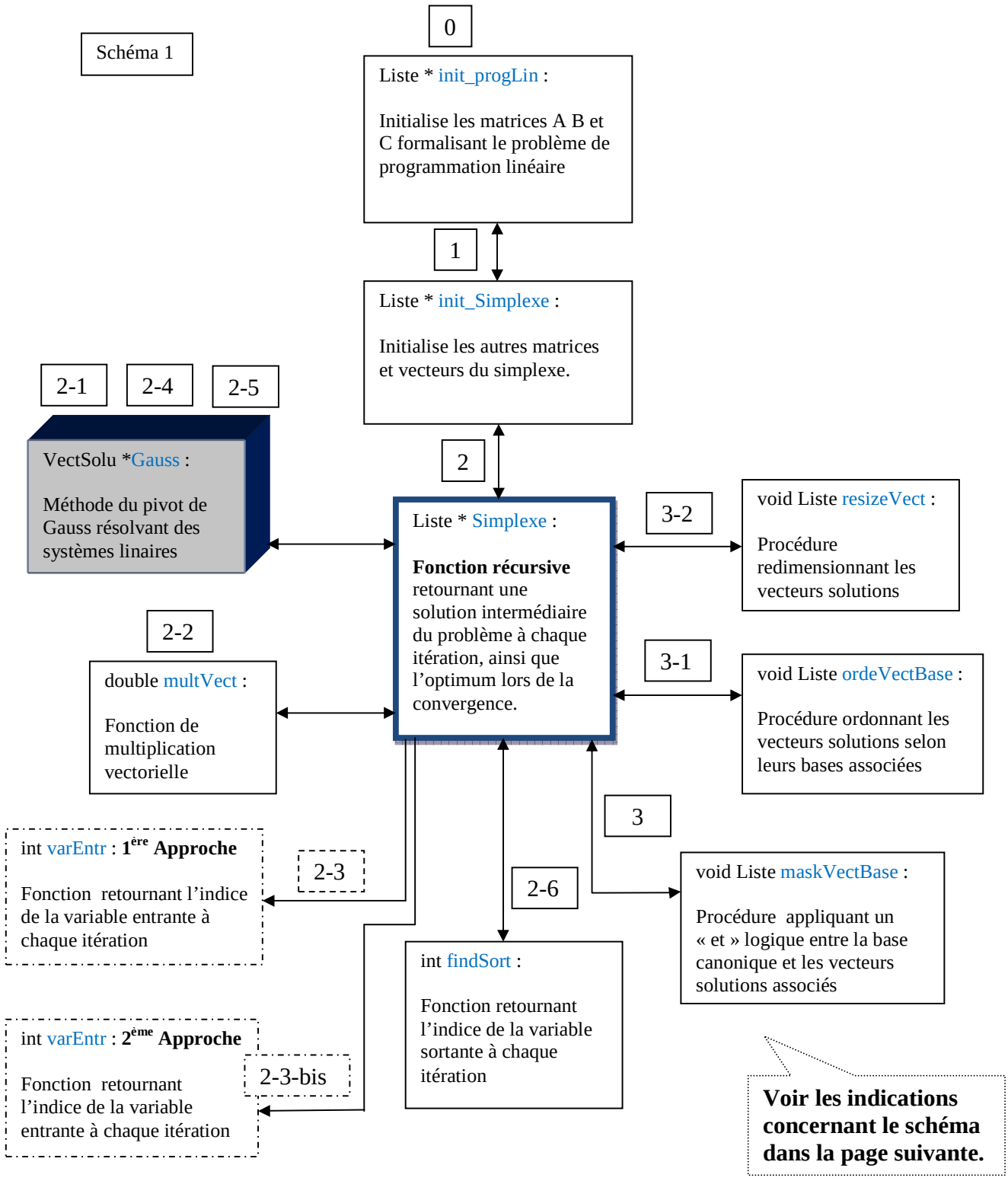
3. On résout : $Bd = a_1$ avec $d = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix} \in \Re^2$

4. Calcul de la variable de sortie en déterminant *t maximum* tel que : $X_B^* - td \geq 0$
   Trouver le plus grand *t* tel que $X_B^* - td \geq 0$. S'il n'existe pas de telle valeur de *t*, le problème est non borné. Sinon, lorsque *t* atteint cette valeur maximum, une au moins des composantes de $X_B^* - td$ vaut 0 et la variable correspondante peut quitter la base, on dira d'elle sortante.

5. Actualisation : Remplacer dans X_B la variable sortante par la variable entrante. Mettre à *t* dans X_B^* la valeur de la variable entrante, et pour les autres variables remplacer leur valeur dans $X_B^* - td$. Dans *B*, remplacer la colonne sortante, c'est-à-dire la colonne associée à la variable sortante, par la colonne entrante.

REMARQUE. Lorsque le problème est donné sous forme standard, si la solution nulle est réalisable, on peut initialiser l'algorithme avec $B=I$ et $X_B^*=b$, ceci correspond au choix de l'ensemble des variables d'écart pour constituer la première base.

3.7 Implémentation : Schéma de fonctionnement du code source



3.7.1 Indications sur le schéma

La numérotation indique les étapes d'exécution du programme Simplexe.cpp. Ce dernier retourne une liste contenant tous les vecteurs solutions intermédiaires, ainsi que l'optimum en dernière instance.

0. Remplissage de la matrice **A** et des vecteurs **b** et **c** formalisant le problème de programmation linéaire

1. Initialisation des matrices A_N, B et des vecteurs $X_N, X_B, X_B^*, C_N, C_B$ à partir des paramètres A, b, et c.

2. Appel à la fonction **récursive Simplexe,** utilisant les fonctions auxiliaires comme suit**:**

- 1ère Appel à la fonction Gauss pour résoudre $y.B = c_B$ lors de l'étape 2-1
- Calcul des variables entrantes potentielles $y.a_j < c_j$ à l'étape 2-2 grâce à multVect
- Calcul de la meilleure variable entrante grâce à varEntr à l'étape 2-3 ou 2-3-bis
- 2ème Appel à Gauss pour résoudre $Bd = a_j$ à l'étape 2-4
- 3ème Appel à Gauss pour trouver le t maximum tel que : $X_B^* - td \geq 0$ à l'étape 2-5
- Retour de la variable sortante via findSort à l'étape 2.6, et grâce aussi à 2-4 et 2-5

3. Les dernières étapes : 3, 3-1, et 3-2 servent à traiter la liste des vecteurs retournée, afin des les retourner écrits dans la base canonique.

3.7.2 Orientation objet

Afin de pouvoir afficher les solutions intermédiaires sur l'interface graphique, il est nécessaire de les retourner de façon adéquate. L'idée consiste à construire une liste au fur et a mesure du déroulement du programme. Pour ce faire, on s'appuie sur les objets suivants :

- vectSolu: Objet contenant un vecteur solution.
- nœud: Objet empilant un vectSolu ainsi que sa base associée.
- Liste: Objet se constituant d'une suite de nœud.

3.7.3 Langage d'implémentation

Pour l'implémentation de la méthode matricielle du simplexe, nous avons opté pour le langage C++ en raison de ces qualités de portabilité, ainsi que par rapport à possibilité d'utiliser différents niveaux d'optimisation au sein d'un même programme (objets - langage structuré classique).

3.7.4 Pivotage : Première approche

Dans le paragraphe consacré à la présentation du principe de l'algorithme (cf.5.2), nous avons expliqué que la convergence de l'algorithme se produit lors de la sélection d'un sommet optimisant au mieux l'objectif. Les règles de sélection sont appelées règle de pivotage, il en existe plusieurs.

Théorème de bland. *Il est dit qu'il ne peut y avoir cyclage lorsque, á toute itération effectuée á partir d'un dictionnaire dégénéré, on choisit les variables entrante et sortante comme celles du plus petit indice parmi les candidats possibles.*

Dans la première approche, on applique le théorème ci-dessus. On choisit à chaque itération parmi les variables entrantes candidates, celle dont l'indice est le plus petit.

Cette méthode appliquée au problème linéaire dans $\Re^2$ retourne en règle générale l'optimum de la fonction objectif. Elle reste aussi valable pour certains cas dans $\Re^3$. Néanmoins, durant les tests réalisés, nous nous sommes aperçus de l'arrêt du processus, à un sommet près du sommet retournant l'optimum de la fonction objectif. Certes, le résultat reste assez probant, car très proche de la valeur maximum (minimum) selon qu'on maximise (minimise), cependant, il est possible de l'améliorer.

3.7.5 Pivotage : Deuxième approche

Cette nouvelle approche a été implémentée dans le but d'améliorer les résultats de l'approche précédente, plus précisément concernant les cas dans $\Re^2$.

Dans cette approche, on choisit parmi les variables entrantes candidates, celle dont le coefficient dans la fonction objectif est le plus fort (le plus faible), selon qu'on maximise (minimise). Cette nouvelle règle de pivotage, nous a permis d'améliorer nos résultats dans la majorité des cas de problèmes linéaires dans $\Re^3$. Toutefois, les différents tests nous ont amené à conclure que cette méthode est moins efficace que la première dans la plupart des cas dans $\Re^2$, sauf pour les problèmes non-bornés (cf. Annexe).

3.7.6 Constat sur les deux approches

Nous avons relevé à travers tous les tests effectués (cf. Annexe), les points suivants (indépendamment du fait que le problème soit dans $\Re^2$ ou $\Re^3$) :

- Si la 1ère approche converge vers le sommet optimum, la 2ème converge vers un de ces sommets adjacent, et **inversement**.
- Les deux approches convergent vers le même sommet optimum, si le problème est non-borné.

4 Interface graphique deux dimensions

Dans cette partie, nous allons aborder la représentation graphique de la résolution d'un programme linéaire dans $\Re^2$, par la méthode du simplexe.

4.1 Présentation

La figure 1 ci-dessous à gauche représente la forme graphique d'un problème de programmation linéaire :

- Les contraintes sont représentées par des droites (indiquées par des flèches)
- L'enveloppe convexe est représentée par un polygone (colorée en gris)

La figure 2 illustre la correspondance avec la résolution du problème d'optimisation, par la méthode du simplexe :

- Le déplacement du pivot est indiqué par des traits rouges
- Les solutions intermédiaires sont mises en évidence par des ronds verts.

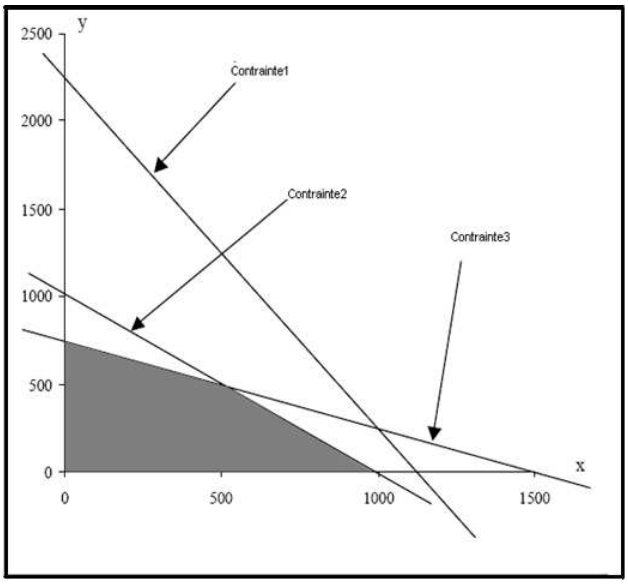


Figure 1 : Forme graphique d'un PL

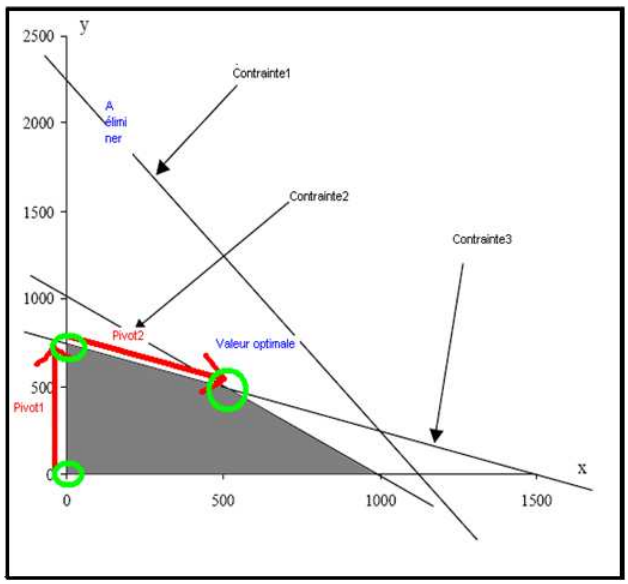


Figure 2 : Résolution du PL par le simplexe

Note. Un problème de programmation linéaire est abrévié par la réduction PL.

4.2 Etapes de mise en œuvre

La réalisation de l'application représentant la figure 2 dans son ensemble comprend plusieurs étapes, explicitées dans les paragraphes ci-dessous.

4.2.1 Choix de la bibliothèque graphique

Qt est une bibliothèque logicielle orientée objet et développée en C++. Elle offre des composants d'interface graphique et d'accès aux données. Nous avons opté pour **Qt**, en raison de ses qualités de portabilité. Les environnements supportés sont les Unix (dont Linux) qui utilisent le système graphique X Window System, Windows et Mac OS X.

4.2.2 Représentation des contraintes

Pour le calcul de l'enveloppe convexe, on commence par dessiner les droites représentant les contraintes. Le schéma obtenu pour deux contraintes quelconques est de la forme ci-dessous.

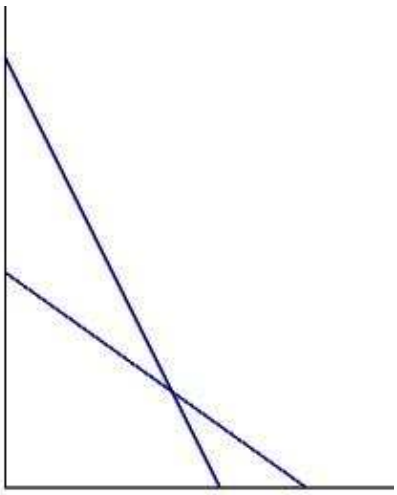


Figure 3 : Représentation graphique de deux inégalités linéaires

4.2.3 Détermination des points d'intersections

On détermine les intersections des droites entre elles, en résolvant un système de 2 équations à 2 inconnues. Ainsi que leurs intersections avec les deux axes (abscisses, ordonnées).

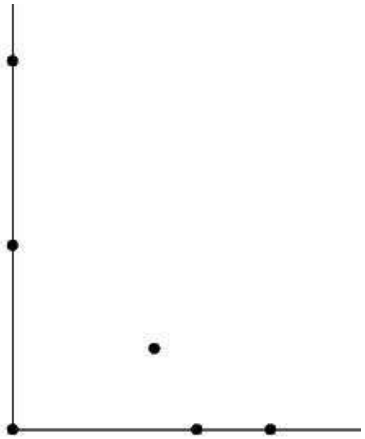


Figure 4 : Représentation des points d'intersections

4.2.4 Suppression des points en dehors de l'enveloppe convexe

On supprime tous les points qui ne vérifient pas toutes les contraintes.

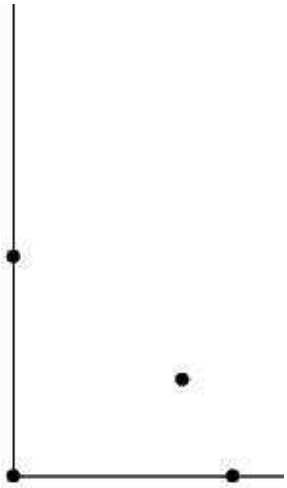


Figure 5 : Suppression des points en dehors de l'enveloppe

4.2.5 Représentation de l'enveloppe convexe : Polygone

Il ne reste plus qu'à joindre les points présents dans le plan, mais dans le bon ordre. Cette notion d'ordre sera abordée ultérieurement.

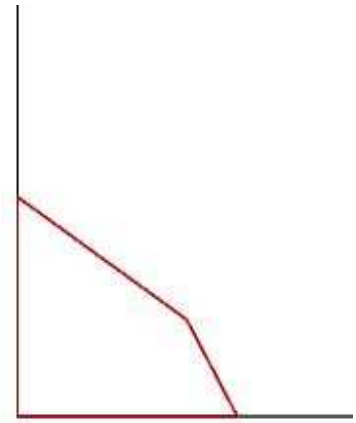
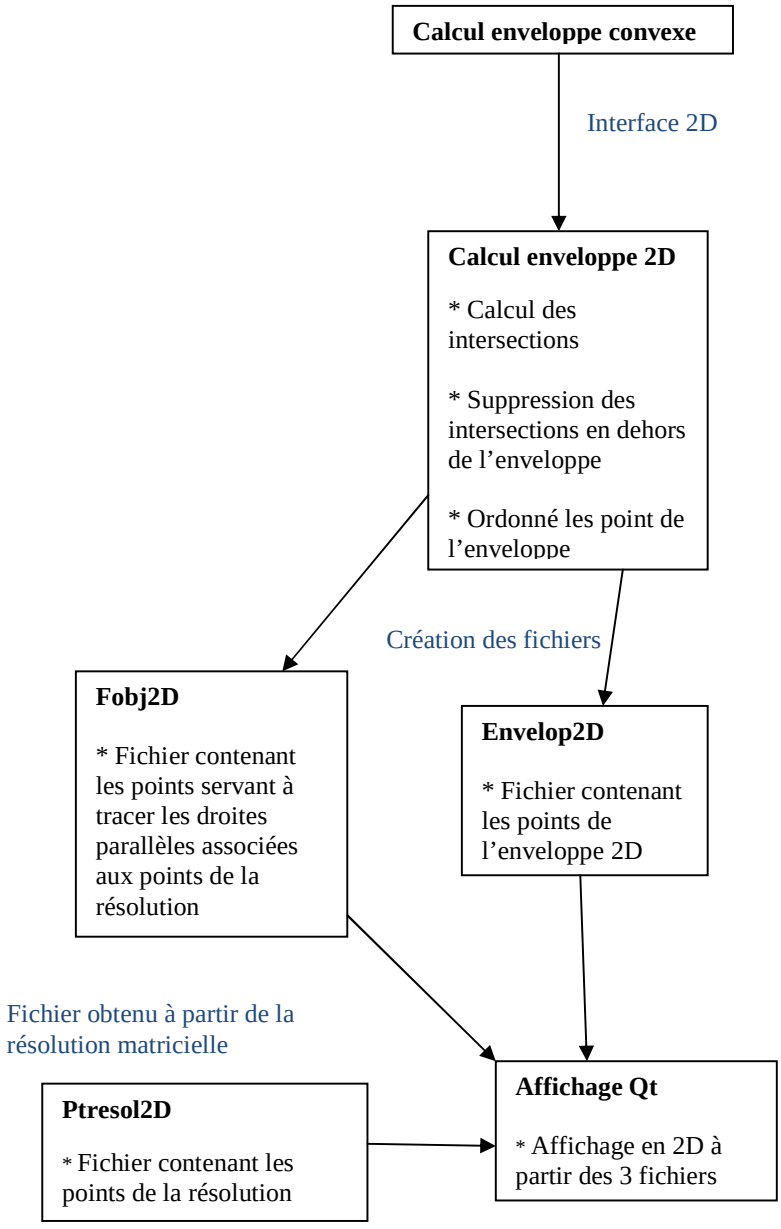


Figure 6 : Dessin de l'enveloppe convexe

4.3 Implémentation : Schéma de fonctionnement du code source

Schéma 2



Note. Les points de la résolution représentent les solutions intermé--diaires, contenus dans la liste retournée par le simplexe (cf.4.7.2).

Voir les indications concernant le schéma dans la page suivante.

4.3.1 Indications sur le schéma

On crée des objets « point2D » qui représentent les points. Ils sont caractérisés par les attributs « x », « y ».

On représente les contraintes avec les objets « contrainte2D ». Ils ont les attributs « cx », « cy », « cst », et « sign ». Les attributs « cx », « cy » et « cz » sont les coefficients des variables de la contrainte, « cst » est l'élément constant et « sign » un nombre qui représente l'opérateur de la contrainte (1 pour <= et 2 pour >=).

Exemple de contraintes : cx*x1+cy*x2 <= cst.

La fonction « calculenv2D » prend respectivement en paramètre un vecteur « contrainte2D », et retourne un vecteur de « point2D » qui représente les points de l'enveloppe convexe.

Cette fonction va faire appel dans un premier temps à « resolution2D » qui prend en paramètre un vecteur de « contrainte2D » et retourne un vecteur de « point2D » qui représente les intersections des droites associés aux contraintes.

Dans un deuxième temps, pour chacun des éléments du vecteur obtenu ci-dessus (autrement dit, pour chacune des intersections) on ne va garder que ceux qui vérifient toutes les contraintes (cela revient à enlever tous les points qui n'appartiennent pas à l'enveloppe convexe) avec les fonctions « verifcontrainte2D » (qui renvoie vrai si le point passé dans le 1er paramètre vérifie toutes les contraintes passées dans le 2nd) et on obtient alors l'enveloppe convexe.

4.3.2 Traitement des données pour l'affichage

Pour pouvoir tracer un polygone, le support Qt requiert les sommets du polygone dans l'ordre. En effet, il est possible de dessiner plus d'une forme géométrique à partir d'un nombre de sommets strictement supérieur à 3.

Exemple. La figure ci-dessous, illustre les deux formes possibles obtenues à partir de quatre sommets donnés.

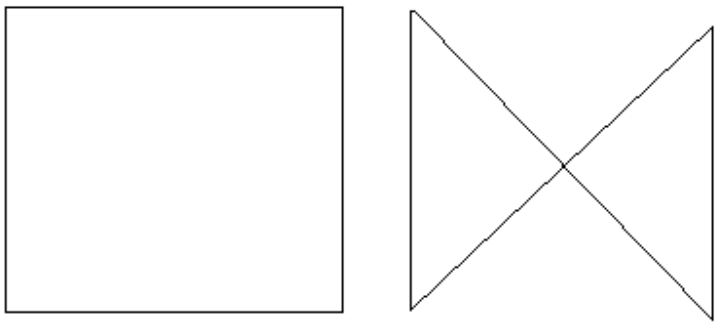


Figure 7 :

Formes possibles à partir de quatre sommets donnés

Il est donc impératif d'ordonner les points afin d'obtenir la forme adéquate. Pour ce faire, nous utilisons la fonction « ordonnept2D » (cf. Schéma 2) qui va à partir du point avec la plus petite ordonnée et la plus petite abscisse, calculer les angles polaires par rapport aux autres points, et les ordonner de façon croissante. La figure ci-dessous explicite l'opération.

Sur l'exemple ci-contre, en ordonnant les points par ordre croissant de leur angle polaire, à partir du point 0, on obtient la suite de points 0, 1, 2 et 3.

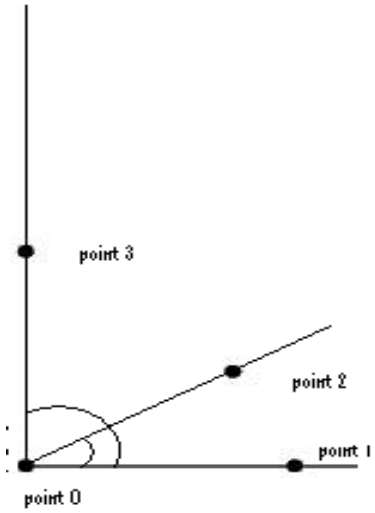


Figure 8 :

Ordonnancement adéquat des points

4.3.3 Résultat de l'affichage 2D sous Qt

On récupère à partir des trois fichiers **Fobj2D, Ptresol2D et Envelop2D** (cf. Schéma 2) respectivement :

- Les points de l'enveloppe convexe
- Les points des solutions intermédiaires obtenues par la résolution (indiqués en rouge ci-dessous)
- Les points servant à tracer les droites parallèles associées au point de la résolution (indiqués en bleu).

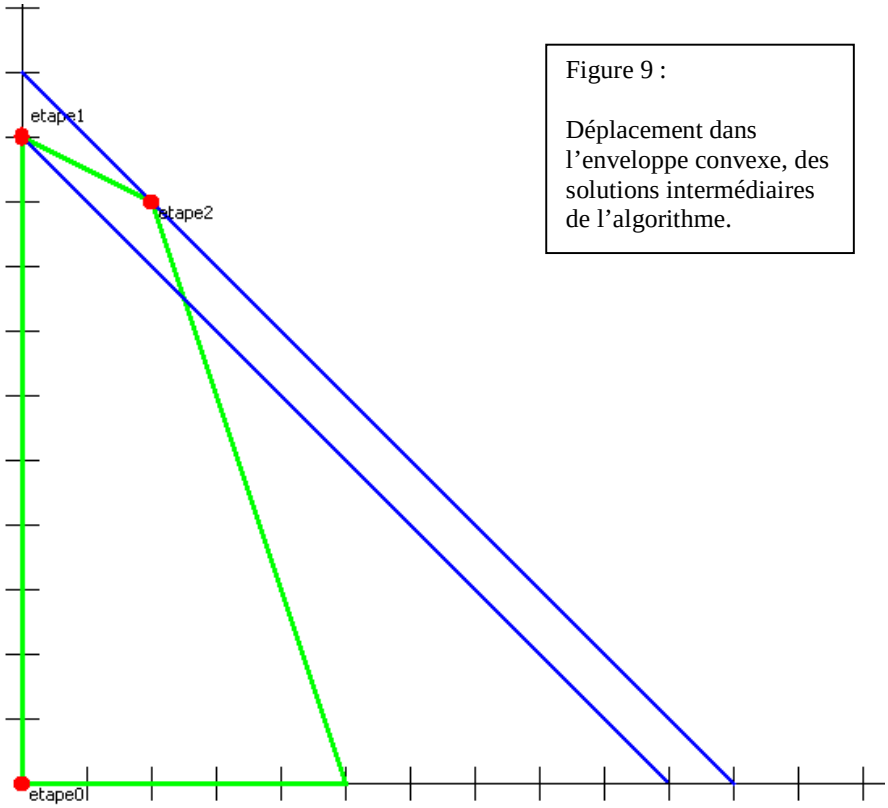


Figure 9 :

Déplacement dans l'enveloppe convexe, des solutions intermédiaires de l'algorithme.

Note. On a utilisé des fichiers extérieurs car par défaut, les exécutables Qt n'utilisent pas de console et cela pose problème pour les saisies. C'est pourquoi, à la fin du programme on fait un appel système qui appel l'exécutable du programme en Qt et le lance. Ce programme récupère alors les données à partir des fichiers et les affiche.

5 Interface graphique trois dimensions

5.1 Préliminaires

Dans cette partie, nous allons aborder la représentation graphique de la résolution d'un programme linéaire dans $\Re^3$, par la méthode du simplexe. La réalisation de l'application comprend plusieurs étapes, explicitées dans les paragraphes ci-dessous.

Le sous paragraphe ci-dessous est séparé des autres étapes de mise en œuvre, car il repose sur les mêmes mécanismes établis dans le cas 2D.

5.1.1 Récupération des points de l'enveloppe : Polyèdre

Note. Les contraintes dans le cas 3D sont représentées par des plans.

Pour le calcul de l'enveloppe convexe, on procède suivant la même logique énoncée dans le cas 2D. Les étapes que comporte cette opération sont explicitées ci-dessous :

- On détermine les intersections des plans (contraintes) avec les plans du repère orthonormé 3D.
- On détermine les intersections des plans (contraintes) entre eux, en résolvant un système de 3 équations à 3 inconnues.
- On supprime tous les points qui ne vérifient pas toutes les contraintes.

5.2 Etapes de mise en œuvre :

5.2.1 Choix de la bibliothèque graphique

On a choisit OpenGL (***Open Graphics Library***). C'est une API multiplateforme dédiée à la conception des applications générant des images 3D (également 2D).

OpenGL regroupe environ 250 fonctions différentes qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes, à partir de simples primitives géométriques.

Notre choix de cette API a été motivé par sa souplesse d'utilisation, ainsi que sa disponibilité sur toutes les plateformes. Par ailleurs, elle est utilisée par la majorité des applications 3D scientifiques, industrielles et artistiques.

5.2.2 Segmentation de l'enveloppe convexe

Note. Il est nécessaire de procéder à cette segmentation, car le logiciel qu'on a choisi pour la réalisation de l'interface, requiert pour la représentation d'un polyèdre, les points appartenant à chacune de ses facettes.

Le but de cette opération est de déterminer les sommets appartenant à chacune des faces du polyèdre. Pour ce faire, nous déterminons autant d'ensembles E_i que de contraintes C_i. Ensuite, on joint à chaque ensemble E_i , tous les points de l'enveloppe convexe (déterminés dans l'étape précédente. cf. 6.2.1) qui vérifient la contrainte C_i.

5.2.3 Ordonnancement des facettes de l'enveloppe convexe

Nous disposons à partir des deux étapes précédentes (cf.6.1.1 et 6.1.2) de tous les sommets du polyèdre. De plus, segmentés. Chacun des segments contient les points appartenant à une seule des facettes du polyèdre. Cependant, ils ne sont pas ordonnés. Ainsi, comme on l'a vu dans le cas 2D (cf.5.3.3), la forme adéquate de l'enveloppe convexe n'est obtenue que si l'ordonnancement des points dans l'espace est correct. Nous allons donc procéder à l'ordonnancement des points, segment par segment, autrement dit, dans chacune des facettes.

Afin d'ordonner les points, l'idée dans le cas 2D consistait à transformer les coordonnées cartésiennes en coordonnées polaires, ***(r,θ)***, puis de trier les points par rapport à l'angle ***θ***, que forme l'arête qui relie ce point à l'origine avec l'axe des ***X***.

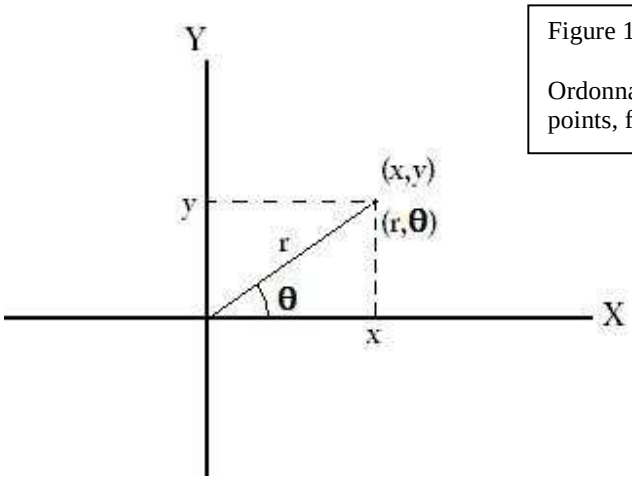
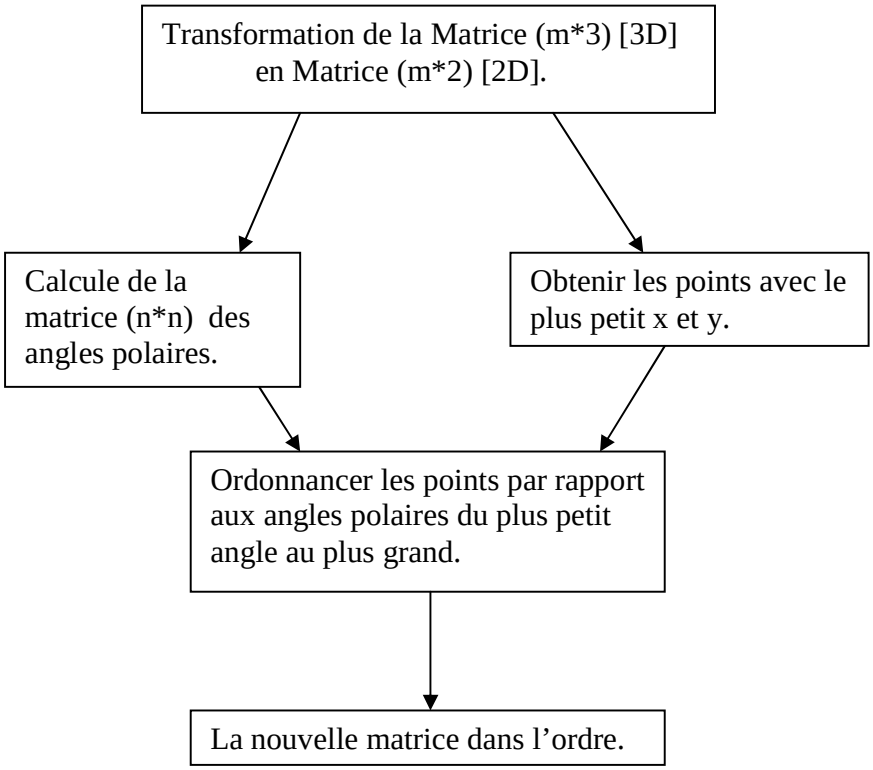


Figure 10 :

Ordonnancement des points, facette par facette.

Sur un repère 3D, on a trois coordonnées cartésiennes, le problème est donc de savoir comment trier les points de chacune des facettes et selon quelles coordonnées. Pour ce faire, nous utilisons les coordonnées polaires avec le même principe que dans la 2D, mais en éliminant la troisième coordonnée. On travaille donc sur les 2 coordonnées **(x, y)** qui restent. L'idée consiste donc à faire une simple projection de toute la facette sur le plan **(z = 0)**. Cette méthode retourne les résultats escomptés, sauf dans les cas où on a deux points qui ont la même abscisse (**x)**, ainsi que la même coordonnée (**y)**, mais évidemment pas le même **z**. Ceux ne sont pas les mêmes points, mais avec la projection en question, ils le deviennent. Pour résoudre ce problème on permute entre le **y** et le **z**.

Le schéma suivant représente la fonction *tri* qui ordonnance les points :



5.2.4 Utilisation des primitives dans OpenGL

Désormais, nous disposons des sommets appartenant à chacune des facettes. Pour réaliser la représentation graphique, il faut utiliser les primitives déjà présentes dans OpenGL, en particulier [***glVertex3d(x,y,z)***]. Celle-ci prend en paramètre les coordonnées cartésiennes d'un point, et les placent directement sur le plan. Une succession de cette primitive fait automatiquement le lien entre les points qu'elle construit (arêtes entre les points). La succession de primitives doit être comprise entre deux clauses [***glBegin(***type de dessin souhaité***)***] et [***glEnd()***].

Les types de dessins mentionnés dans la fonction [***glBegin()***] sont :

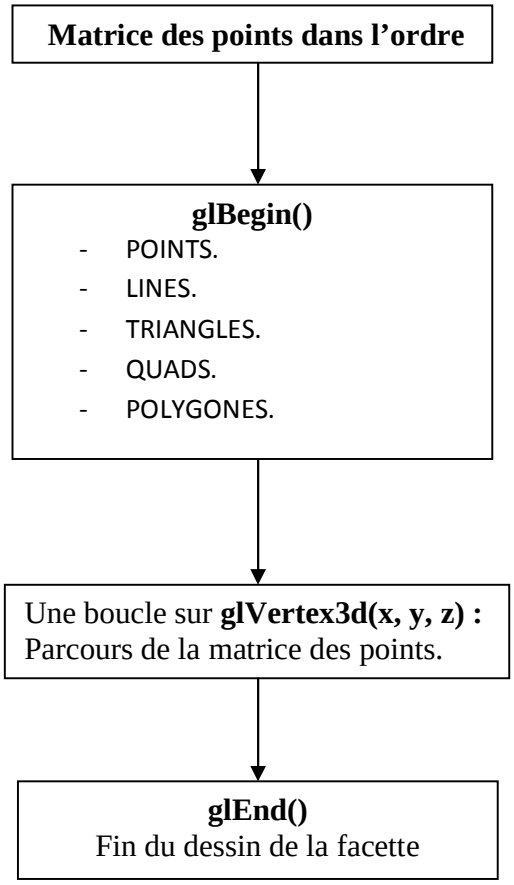
- Pour les formes vides :

  ⇨ [***glBegin(GL_POINTS)***] construit simplement des points
  ⇨ [***glBegin(GL_LINES)***] construit des lignes entre les points (arêtes)

- Pour les formes pleines :

  ⇨ [***glBegin(GL_TRIANGLES)***] construit des facettes de trois points (*Triangle*)
  ⇨ [***glBegin(GL_QUADS)***] construit des facettes à quatre points (*Quadrilatère*)
  ⇨ [***glBegin(POLYGON)***] pour des facettes à plus de quatre points (*Pentagone*)

A l'aide de ces primitives, on représente la forme 3D correspondante à l'enveloppe convexe du problème linéaire. Cette opération est effectuée par la fonction *dessiner* dont le schéma est le suivant :



5.2.5 Eviter la superposition des facette

Lors des premiers essais sur OpenGL, la superposition des faces des objets a très rapidement posée problème.

Cas simple. Dans le cas d'un cube, il faut éviter que les facette qui sont en arrière plan apparaissent.

La résolution de ce problème réside dans l'utilisation de l'algorithme **Z-Buffer**. Il permet d'élimination les parties cachées. Ainsi, lors de la visualisation d'une scène, cet algorithme supprime l'affichage de tout objet ou morceau d'objet masqué par un autre objet ou par lui-même.

Principe. Le calcul de l'image est effectué séquentiellement en traitant les objets extraits de la scène (classiquement des facettes) les uns à la suite des autres pour en extraire l'ensemble des pixels qui les représentent. Au cours du processus de traitement d'un objet, chaque pixel calculé voit sa profondeur comparée à celle déjà calculée en cette place pour pouvoir n'afficher finalement pour chaque pixel de l'image que la facette la moins profonde présente en ce pixel (vis à vis de l'observateur).

Il ne s'agit pas d'un tri, mais un calcul de maximum pour chaque pixel vis à vis de l'ensemble des objets présents en ce pixel.

5.2.6 Rotation des objets

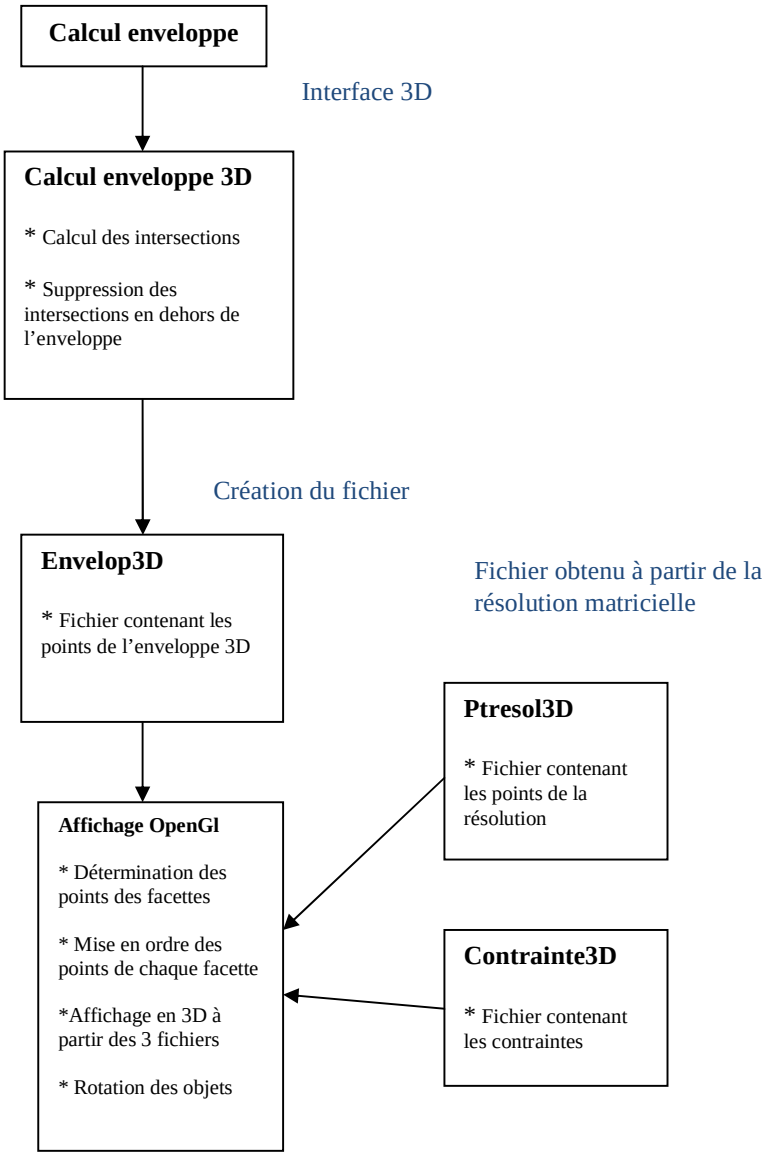
Il est necessaire d'avoir la possiblité de voir l'objet de tous ces cotés, car les problèmes linéaires ne sont pas tous de forme idéale (**Ex.** cube, pyramide). Pour ce faire, il existe deux manières de procéder :

- En faisant des rotation de l'objet sur les 2 axes ***(Ox,Oz)***
- En jouant sur l'angle de la caméra

On a opté pour la rotation des objets, par rapport aux axes. On a fixé une vitesse optimale et on a laissé l'objet en mouvement libre qui permet de visualiser toutes les faces. Cette rotation est réalisée avec la primitive [***glRotated(angleX,1,0,0)***]. Cette primitive est un exemple de rotation sur l'axe des ***x***, avec un ***angleX***, on incrémente à chaque fois l'***angleX***, ce qui fait tourner l'objet sur l'axe des ***x***.

5.2.7 Implémentation : Schéma de fonctionnement global

Schéma 3



5.2.8 Indications sur le schéma

On crée des objets « point3D » qui représentent les point, ils sont caractérisés par les attributs « x », « y » et « z » .

On représente les contraintes avec les objets « contrainte3D ». Ils ont les attributs « cx », « cy », « cz » « cst » et « sign ». Les attributs « cx », « cy » et « cz » sont les coefficients des variables de la contrainte, « cst » est l'élément constant et « sign » un nombre qui représente l'opérateur de la contrainte (1 pour <= et 2 pour >=).

Exemple de contraintes : cx*x1+cy*x2+cz*x3<=cst.

Les fonctions « calculenv3D » prend en paramètre un vecteur « contrainte3D » et retournent un vecteur de « point3D » qui représentent les points de l'enveloppe convexe.

Ces fonctions vont faire appel dans un premier temps à « resolution3D » qui prend en paramètre un vecteur de « contrainte3D » et retourne un vecteur de « point3D » qui représentent les intersections des plans associés aux contraintes.

Dans un deuxième temps, comme pour la 2D, on prend chacun des éléments du vecteur obtenu ci-dessus (autrement dit, pour chacune des intersections) on ne va garder que ceux qui vérifient toutes les contraintes (cela revient à enlever tous les points qui n'appartiennent pas à l'enveloppe) avec les fonctions « verifcontrainte3D » (qui renvoie vrai si le point rentrer en paramètre vérifie toute les contraintes rentrées en paramètre) et on obtient notre enveloppe convexe.

5.2.9 Résultat de l'affichage 3D sous OpenGL

La sphère bleue est une solution intermédiaire. La rouge représente le point de convergence, donc la deuxième et dernière solution

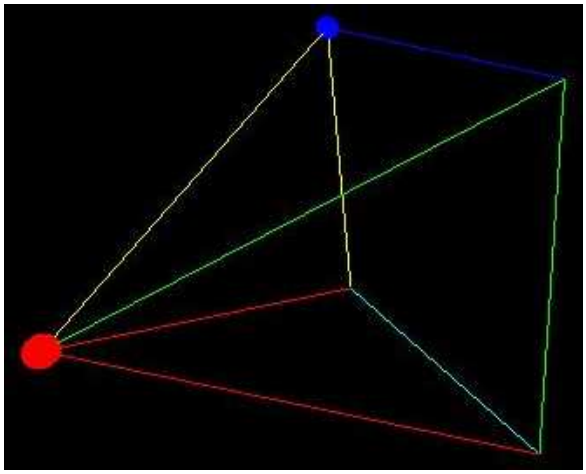


Figure 11 : Représentation d'une pyramide vide sous OpenGL.

6 Bilan de l'étude

Les objectifs déterminés dans le **C**ahier **D**es **C**harges ont été atteints dans leur ensemble. Les trois opérations fondamentales du projet (cf.2.2) ont été ont été réalisées avec succès. Nous rappelons ci-dessous les principaux points concrétisés :

- Implémentation de l'algorithme du simplexe

⇨ Résolution des problèmes de programmation linéaires dans $\Re^n$
⇨ Identification de toutes les solutions intermédiaires
⇨ Convergence vers l'optimum de la fonction objectif
⇨ Ecriture des solutions dans la base canonique du problème

- La réalisation d'une interface graphique consacrée aux problèmes dans $\Re^2$

⇨ Construction de l'enveloppe convexe : Polygone
⇨ Construction des points associés aux solutions intermédiaires
⇨ Constatation d'un déplacement sur l'enveloppe convexe, par la mise en évidence graphique, des étapes de la résolution par le simplexe.
⇨ Construction des droites parallèles de la fonction objectif, passants par chacune des solutions intermédiaires.

- La réalisation d'une plateforme graphique dédiée aux problèmes dans $\Re^3$

⇨ Construction de l'enveloppe convexe : Polyèdre
⇨ Construction des points associés aux solutions intermédiaires
⇨ Constatation d'un déplacement sur l'enveloppe convexe, par la mise en évidence graphique, des étapes de la résolution par le simplexe.
⇨ Rotation de l'enveloppe convexe dans l'espace 3D.

Les mises en œuvre des trois segments d'opérations ci-dessus, se sont déroulées de façon simultanée, et dans les temps voulus. Nous avons donc respecté le parallélisme des tâches décrit dans le diagramme de Gant (cf. VIII dans le **CDC**), ainsi que leurs délais respectifs.

6.1 Indépendance de fonctionnement

Le point fort avéré de nos travaux réside leur indépendance de fonctionnement. En effet, les trois segments du point précédent (cf.7) fonctionnent de façon complètement indépendante les uns des autres. Ils peuvent donc être réutilisés directement par d'autres applications, à savoir :

- Le simplexe qui peut servir de solveur dans $\Re^n$, et qui traite par ailleurs les problèmes non bornés (Convergence vers une des solutions optimales).

- Les interfaces 2D et 3D pour représenter respectivement des courbes dans le plan $\Re^2$ ou des solides dans $\Re^3$.

6.2 Améliorations envisageables

Nous avons relevé trois améliorations possibles, avec lesquelles on pourrait certainement finaliser cette étude :

- Concernant le simplexe
⇨ Implémenter une méthode de standardisation de problème de programmation linéaire.
⇨ Essayer d'autres méthodes de pivotage, dans la perspective de converger vers l'optimum en suivant le chemin le plus court (minimiser les itérations).

- Concernant l'interface 2D
⇨ Implémenter une méthode de temporisation sur Qt, afin de constater plus agréablement le déplacement des solutions sur l'enveloppe convexe.

- Concernant l'interface 3D
⇨ Implémenter une méthode de translation d'objet sur OpenGL, afin de constater plus agréablement le déplacement sur l'enveloppe convexe.

7 Conclusion

Les travaux de recherche réalisés en amont nous ont apporté énormément de connaissances théoriques, corrélées au sujet de l'étude. La mise en œuvre nous a quant a elle permit d'améliorer nos aptitudes techniques de programmation, aussi d'acquérir de nouvelles compétences, notamment dans le domaine de la représentation graphique.

Par ailleurs, le travail d'équipe nous a plongés dans une optique d'entreprise. En effet, nous étions cernés par des objectifs à réaliser, dans des durées négociées au préalable. Aussi, nous devions rendre compte de l'état d'avancement des travaux à notre encadrant, qui dans ce cadre se retrouvait dans la posture du client. La communication s'est donc avérée prépondérante à la cohésion du groupe ainsi qu'au bon déroulement du projet.

8 Annexe

8.1 Représentation graphique 2D

8.1.1 Exemple 1 : Cas borné, 1ère Approche

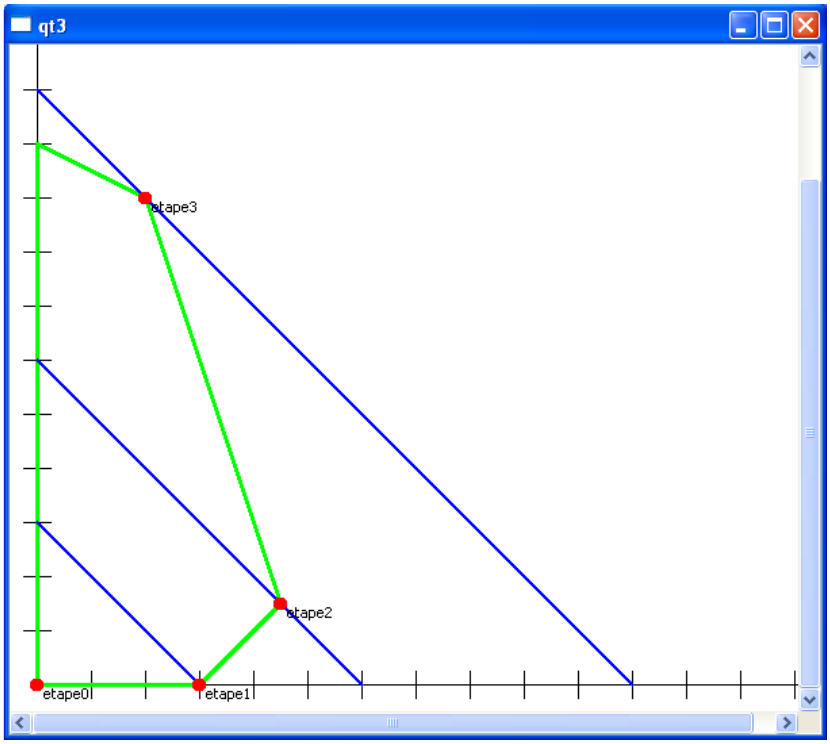
Soit le problème borné dans $\Re^2$ suivant :

$$
\begin{aligned}
&\textit{Maximiser}: \\
&Z = x_1 + x_2 \\
&\textit{avec}: \\
&0.5x_1 + x_2 \leq 10 \\
&3x_1 + x_2 \leq 15 \\
&x_1 - x_2 \leq 3 \\
&x_1, x_2 \geq 0
\end{aligned}
$$

Les résultats sont affichés sur les deux fenêtres ci-dessous :

```
********************************** RESULTATS ***************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
            Ps : Suivis de leurs bases respectives :

8.5 6 3 dans la base : x3 x4 x1

6.25 1.5 4.5 dans la base : x3 x2 x1

10 9 2 dans la base : x5 x2 x1


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 0 3 dans la base : x3 x4 x1

0 1.5 4.5 dans la base : x3 x2 x1

0 9 2 dans la base : x5 x2 x1


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

3 0 0 dans la base : x1 x3 x4

4.5 1.5 0 dans la base : x1 x2 x3

2 9 0 dans la base : x1 x2 x5
```

```
On ecrit les vecteurs dans la base canonique :

3 0   dans la base : x1 x2

4.5 1.5   dans la base : x1 x2

2 9   dans la base : x1 x2


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
       Ps : Ecrits dans la base canonique de |R[2] (from First to Last) :

3 0

4.5 1.5

2 9

************************************ TERMINE ***************************************
```



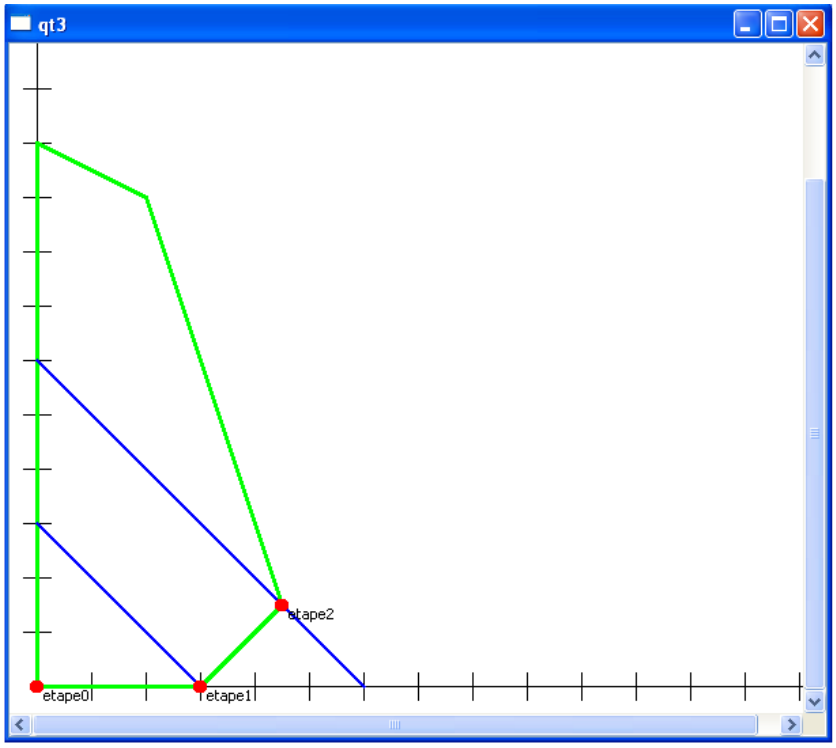
Note. On voit bien sur la représentation graphique ci-dessus, que l'optimum est bien atteint par la première approche.

8.1.2 Suite exemple 1 : 2ème Approche

On applique la deuxième approche au problème précédent :

```
********************************** RESULTATS ****************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
            Ps : Suivis de leurs bases respectives :

8.5 6 3 dans la base : x3 x4 x1

6.25 1.5 4.5 dans la base : x3 x2 x1


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 0 3 dans la base : x3 x4 x1

0 1.5 4.5 dans la base : x3 x2 x1


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

3 0 0 dans la base : x1 x3 x4

4.5 1.5 0 dans la base : x1 x2 x3
```

```
On ecrit les vecteurs dans la base canonique :

3 0 0 dans la base : x1 x2 x4

4.5 1.5 0 dans la base : x1 x2 x3


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[2] (from First to Last) :

3 0

4.5 1.5

************************************ TERMINE ****************************************

userdz@ubuntu:~/U_Directory/TER$
```



Note. On voit bien sur la représentation graphique ci-dessus, que la convergence se produit sur un des sommets adjacent de l'optimum. Ce dernier n'est donc pas atteint par application de la 2ème Approche.

8.1.3 Exemple 2 : Cas non borné, 1ère Approche

Soit le problème non borné dans $\Re^2$ suivant :

Maximiser :

$Z = 2x_1 + 8x_2$

avec :

$-x_1 + x_2 \leq 4$

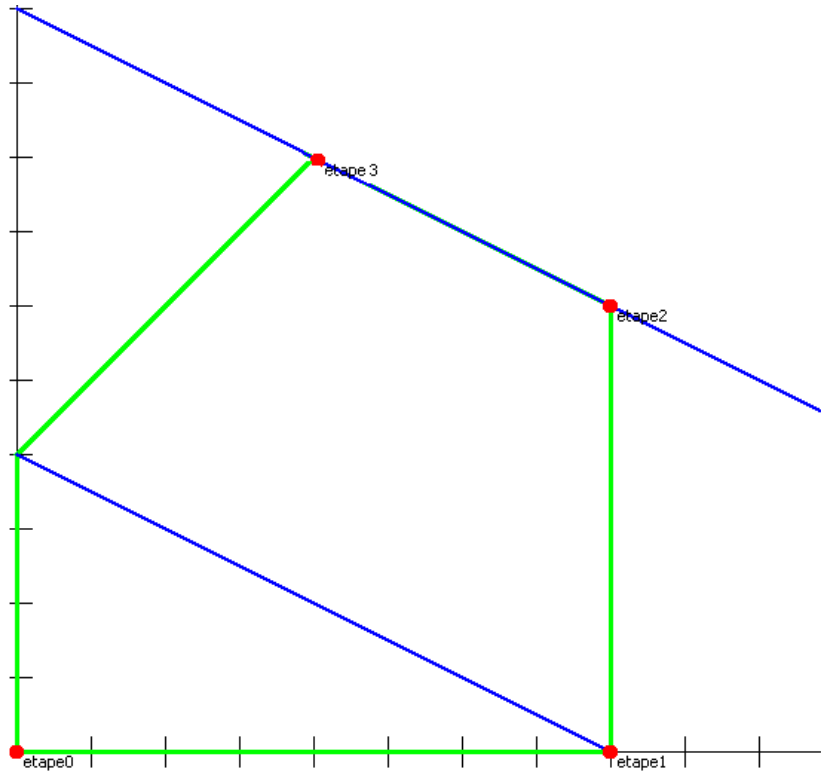
$x_1 + 2x_2 \leq 20$

$x_1 \leq 8$

$x_1, x_2 \geq 0$

```
********************************** RESULTATS ****************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
            Ps : Suivis de leurs bases respectives :

12 12 8 dans la base : x3 x4 x1

6 6 8 dans la base : x3 x2 x1

4 8 4 dans la base : x5 x2 x1


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 0 8 dans la base : x3 x4 x1

0 6 8 dans la base : x3 x2 x1

0 8 4 dans la base : x5 x2 x1


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

8 0 0 dans la base : x1 x3 x4

8 6 0 dans la base : x1 x2 x3

4 8 0 dans la base : x1 x2 x5
```

```
On ecrit les vecteurs dans la base canonique :

8 0   dans la base : x1 x2

8 6   dans la base : x1 x2

4 8   dans la base : x1 x2


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
       Ps : Ecrits dans la base canonique de |R[2] (from First to Last) :

8 0

8 6

4 8

********************************** TERMINE ****************************************
```



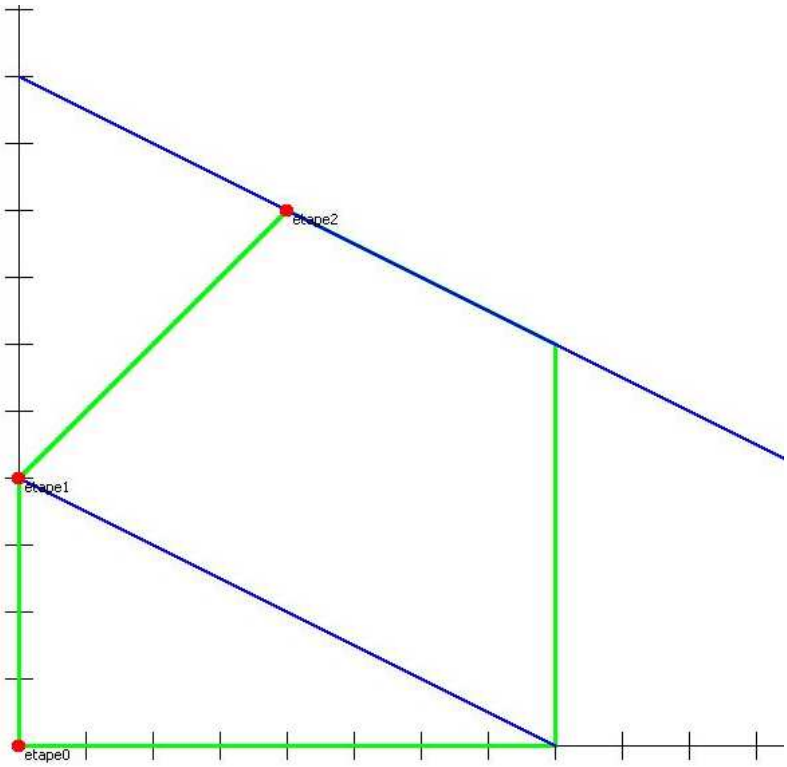
Note. On voit bien sur la représentation graphique ci-dessus, que l'optimum est bien atteint par la première approche.

8.1.4 Suite exemple 2 : 2ème Approche

On applique la deuxième approche au problème précédent :

```
********************************** RESULTATS ***************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
             Ps : Suivis de leurs bases respectives :

4 12 8 dans la base : x2 x4 x5

8 4 4 dans la base : x2 x1 x5


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

4 0 0 dans la base : x2 x4 x5

8 4 0 dans la base : x2 x1 x5


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

4 0 0 dans la base : x2 x4 x5

4 8 0 dans la base : x1 x2 x5
```

```
On ecrit les vecteurs dans la base canonique :

0 4   dans la base : x1 x2

4 8   dans la base : x1 x2


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[2] (from First to Last) :

0 4

4 8

********************************** TERMINE ***************************************

userdz@ubuntu:~/U_Directory/TER$
```



Note. On voit bien sur la représentation graphique ci-dessus, que la convergence se produit comme dans la 1ère Approche sur optimum.

Dans les cas non-bornés, la 2ème approche peut donc être valable, elle est même meilleurs en terme de complexité dans ce cas là, car elle converge en prenant le chemin le plus court.

8.2 Résolution de problèmes 3D selon les approches

8.2.1 Exemple 1 : Résolution théorique

Soit le problème dans $\Re^3$ suivant :

Maximiser :

$$Z = 7x_1 + 9x_2 + 18x_3$$

avec :

$$2x_1 + 4x_2 + 5x_3 \leq 42$$

$$x_1 + x_2 + 2x_3 \leq 17$$

$$x_1 + 2x_2 + 3x_3 \leq 24$$

$$x_1, x_2, x_3 \geq 0$$

1. On introduit une nouvelle variable par contrainte, et le système devient :

$$\left.\begin{array}{rcl} x_4 &=& 1000 - x_1 \\ x_5 &=& 500 - x_2 \\ x_6 &=& 1500 - x_3 \\ x_7 &=& 6750 - 3x_1 - 6x_2 - 2x_3 \end{array}\right\} \quad z = 4x_1 + 12x_2 + 3x_3$$

Les trois variables sont modifiables, et nous choisissons de modifier x_1 (et donc de garder $x_2 = x_3 = 0$). Les contraintes de positivité imposent $x_1 \leq 1000$ $x_1 \leq 2250$. Le meilleur mouvement est donc $x_1 = 1000$, qui conduit au sommet :

$$x_2 = x_3 = x_4 = 0 \quad ; \quad x_1 = 1000,\ x_5 = 500,\ x_6 = 1500,\ x_7 = 3750$$

Itération 1

2. Du point de vue de ce sommet, le système se réécrit :

$$\left.\begin{array}{rcl} x_1 &=& 1000 - x_4 \\ x_5 &=& 500 - x_2 \\ x_6 &=& 1500 - x_3 \\ x_7 &=& 3750 + 3x_4 - 6x_2 - 2x_3 \end{array}\right\} \quad z = 4000 + 12x_2 + 3x_3 - 4x_4$$

Deux variables sont modifiables. Nous choisissons de modifier x_2 (et de ne pas modifier $x_3 = x_4 = 0$). Vu les contraintes de positivité, cette modification doit vérifier $x_2 \leq 500$ et $x_2 \leq 3750/6 = 625$. Le meilleur mouvement possible est donc $x_2 = 500$, qui conduit au sommet :

$$x_3 = x_4 = x_5 = 0 \quad ; \quad x_1 = 1000,\ x_2 = 500,\ x_6 = 1500,\ x_7 = 750$$

Itération 2

3. Du point de vue de ce sommet, le système se réécrit :

$$\left.\begin{array}{rcl} x_1 &=& 1000 - x_4 \\ x_2 &=& 500 - x_5 \\ x_6 &=& 1500 - x_3 \\ x_7 &=& 750 + 3x_4 - 6x_5 - 2x_3 \end{array}\right\} \quad z = 10000 + 3x_3 - 4x_4 - 12x_5$$

correction d'une erreur : ce n'est pas $-6x_5$ mais $+6x_5$

La seule variable modifiables est x_3. Les contraintes de positivité imposent $x_3 \leq 1500$ et $x_3 \leq 375$. Le meilleur mouvement possible est donc $x_3 = 375$, qui conduit au sommet :

$x_4 = x_5 = x_7 = 0$; $x_1 = 1000, x_2 = 500, x_3 = 375, x_6 = 1125$ Itération 3

4. Du point de vue de ce sommet, le système se réécrit :

$$\left.\begin{array}{rcl} x_1 &=& 1000 - x_4 \\ x_2 &=& 500 - x_5 \\ x_3 &=& 375 + \frac{3}{2}x_4 + 3x_5 - \frac{1}{2}x_7 \\ x_6 &=& 1125 - \frac{3}{2}x_4 - 3x_5 + \frac{1}{2}x_7 \end{array}\right\} \quad z = 11125 + \frac{1}{2}x_4 - 3x_5 - \frac{3}{2}x_7$$

La seule variable modifiable est x_4, avec pour contraintes $x_4 \leq 1000$ et $x_4 \leq 1125 \times \frac{2}{3} = 750$. Le meilleur mouvement est donc $x_4 = 750$, conduisant au sommet

$x_5 = x_6 = x_7 = 0$; $x_1 = 250, x_2 = 500, x_3 = 1500, x_4 = 750$ Itération 4

5. Du point de vue de ce sommet, le système se réécrit :

$$\left.\begin{array}{rcl} x_1 &=& 250 + 2x_5 + \frac{2}{3}x_6 - \frac{1}{3}x_7 \\ x_2 &=& 500 - x_5 \\ x_3 &=& 1500 - x_6 \\ x_4 &=& 750 - 2x_5 - \frac{2}{3}x_6 + \frac{1}{3}x_7 \end{array}\right\} \quad z = 11500 - 4x_5 - \frac{1}{3}x_6 - \frac{4}{3}x_7$$

Il ne reste plus de variables modifiables, provoquant l'arrêt de l'algorithme. La décision proposée est donc

$x_1 = 250, x_2 = 500, x_3 = 1500$ CONVERGENCE

qui vérifie toutes les contraintes et conduit à un bénéfice prévisionnel de 11500.

8.2.2 Suite exemple 1 : Résolution par la 1ère Approche

Ci-dessous, la résolution par la 1ère approche :

```
Ci-dessous les vecteurs des solutions intermediaires (dans |R[4]) ;
            Ps : Suivis de leurs bases respectives :

1000 500 1500 3750 dans la base : x1 x5 x6 x7

1000 500 1500 750 dans la base : x1 x2 x6 x7

1000 500 1125 375 dans la base : x1 x2 x6 x3

250 500 750 1500 dans la base : x1 x2 x4 x3


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

1000 0 0 0 dans la base : x1 x5 x6 x7

1000 500 0 0 dans la base : x1 x2 x6 x7

1000 500 0 375 dans la base : x1 x2 x6 x3

250 500 0 1500 dans la base : x1 x2 x4 x3


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

1000 0 0 0 dans la base : x1 x5 x6 x7

1000 500 0 0 dans la base : x1 x2 x6 x7

1000 500 375 0 dans la base : x1 x2 x3 x6

250 500 1500 0 dans la base : x1 x2 x3 x4
```

```
 On ecrits les vecteurs dans la base canonique :

1000 0 0   dans la base : x1 x2 x3

1000 500 0   dans la base : x1 x2 x3

1000 500 375   dans la base : x1 x2 x3

250 500 1500   dans la base : x1 x2 x3


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[3] (from First to Last) :

1000 0 0

1000 500 0

1000 500 375

250 500 1500

************************************ TERMINE ****************************************

userdz@ubuntu:~/U_Directory/TER$
```

Note. La convergence se produit bien sur le sommet réalisant l'optimum de la fonction objectif.

8.2.3 Suite exemple 1 : Résolution par la 2ème Approche

Ci-dessous, la résolution par la 2ème approche :

```
********************************** RESULTATS ***************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[4]) ;
            Ps : Suivis de leurs bases respectives :

1000 500 1500 3750 dans la base : x4 x2 x6 x7

1000 500 1500 750 dans la base : x1 x2 x6 x7

1000 500 1125 375 dans la base : x1 x2 x6 x3


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 500 0 0 dans la base : x4 x2 x6 x7

1000 500 0 0 dans la base : x1 x2 x6 x7

1000 500 0 375 dans la base : x1 x2 x6 x3


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

500 0 0 0 dans la base : x2 x4 x6 x7

1000 500 0 0 dans la base : x1 x2 x6 x7

1000 500 375 0 dans la base : x1 x2 x3 x6
```

```
On ecrit les vecteurs dans la base canonique :

0 500 0   dans la base : x1 x2 x3

1000 500 0   dans la base : x1 x2 x3

1000 500 375   dans la base : x1 x2 x3


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[3] (from First to Last) :

0 500 0

1000 500 0

1000 500 375

********************************** TERMINE ***************************************

userdz@ubuntu:~/U_Directory/TER$
```

Note. On voit bien ci-dessus, que la convergence se produit sur un des sommets adjacent de l'optimum. Ce dernier n'est donc pas atteint par application de la 2ème Approche.

8.2.4 Exemple 2 : Résolution par le solveur Excel

Soit le problème dans $\Re^3$ suivant :

Maximiser :

$Z = 7x_1 + 9x_2 + 18x_3$

avec :

$2x_1 + 4x_2 + 5x_3 \leq 42$

$x_1 + x_2 + 2x_3 \leq 17$

$x_1 + 2x_2 + 3x_3 \leq 24$

$x_1, x_2, x_3 \geq 0$

B10 fx =7*B9+9*C9+18*D9

	A	B	C	D	E	F	G
1	Maximiser z = 7X1+9X2+18X3						
2	CONTRAINTES:						
3	Contrainte 1: 2X1+4X2+5X3 <= 42						
4	Contrainte 2: X1+X2+2X3 <= 17						
5	Contrainte 3: X1+2X2+3X3 <= 24						
6	Xi>=0						
7							
8	Variables	X1	X2	X3	X4	X5	X6
9	Valeurs	3	0	7	1	0	0
10	Max z	147					
11							
12			Valeur à atteindre				
13	Contrainte 1	42	42				
14	Contrainte 2	17	17				
15	Contrainte 3	24	24				

8.2.5 Suite exemple 2 : Résolution par la 1ère Approche

Ci-dessous, les résultats de la 1ère approche. On ne converge pas vers l'optimum.

```
********************************** RESULTATS ***************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
            Ps : Suivis de leurs bases respectives :

8 17 7 dans la base : x4 x1 x6

4 13 3 dans la base : x2 x1 x6

1 4 6 dans la base : x2 x1 x3


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 17 0 dans la base : x4 x1 x6

4 13 0 dans la base : x2 x1 x6

1 4 6 dans la base : x2 x1 x3


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

17 0 0 dans la base : x1 x4 x6

13 4 0 dans la base : x1 x2 x6

4 1 6 dans la base : x1 x2 x3
On ecrit les vecteurs dans la base canonique :

17 0 0 dans la base : x1 x2 x3

13 4 0 dans la base : x1 x2 x3

4 1 6 dans la base : x1 x2 x3


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
       Ps : Ecrits dans la base canonique de |R[3] (from First to Last) :

17 0 0

13 4 0

4 1 6

********************************** TERMINE *****************************************

userdz@ubuntu:~/U_Directory/TER$
```

À la convergence, on a :
$(x_1, x_2, x_3) = (4, 1, 6)$.
D'où : **Z = 145**.

8.2.6 Suite exemple 2 : Résolution par la 2ème Approche

Ci-dessous, les résultats de la 2ème approche. On converge cette fois vers l'optimum.

```
************************************ RESULTATS ****************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[3]) ;
            Ps : Suivis de leurs bases respectives :

2 1 8 dans la base : x4 x5 x3

1 3 7 dans la base : x4 x1 x3


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 0 8 dans la base : x4 x5 x3

0 3 7 dans la base : x4 x1 x3


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

8 0 0 dans la base : x3 x4 x5

3 7 0 dans la base : x1 x3 x4

On ecrit les vecteurs dans la base canonique :

0 0 8 dans la base : x1 x2 x3

3 0 7 dans la base : x1 x2 x3


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
       Ps : Ecrits dans la base canonique de |R[3] (from First to Last) :

0 0 8

3 0 7

************************************ TERMINE ****************************************

userdz@ubuntu:~/U_Directory/TER$
```

À la convergence, on a :
$(x_1, x_2, x_3) = (3, 0, 7)$.
D'où : **Z = 147**.

8.3 Résolution d'un problème 4D selon les approches

8.3.1 Exemple : Résolution par le solveur Excel

Soit le problème dans $\Re^4$ suivant :

Maximiser :

$$Z = 5x_1 + 8x_2 + 12x_3 + 15x_4$$

avec :

$$2x_1 + 4x_2 + 5x_3 + 7x_4 \leq 42$$

$$x_1 + x_2 + 2x_3 + 2x_4 \leq 17$$

$$x_1 + 2x_2 + 3x_3 + 3x_4 \leq 24$$

$$x_1 + 2x_2 \leq 5$$

$$x_1, x_2, x_3 \geq 0$$

B11 : =5*B10+D148*C10+12*D10+15*E10

	A	B	C	D	E	F	G	H	I
1	Maximiser nouvelle fonction z = 5X1+ 8X2+12X3+15X4 (l'ancienne c'était z = 7X1+9X2+18X3+17X4)								
2	CONTRAINTES:								
3	Contrainte 1: 2X1+4X2+5X3+7X4+X5 = 42								
4	Contrainte 2: X1+X2+2X3+2X4+X6=17								
5	Contrainte 3: X1+2X2+3X3+3X4+X7=24								
6	Nouvelle contrainte 4 ajoutée: X1+ 2X2 + X8 = 5								
7	Xi>=0								
8									
9	Variables	X1	X2	X3	X4	X5	X6	X7	X8
10	Valeurs	3	0	6,5	0,5	0	0	0	0
11	Max z	100,5							
12									
13			Valeur à atteindre						
14	Contrainte 1	42	42						
15	Contrainte 2	17	17						
16	Contrainte 3	24	24						
17	Contrainte 4	3	5						

8.3.2 Suite exemple : Résolution par la 1ère Approche

Ci-dessous, les résultats de la 1ère approche. On ne converge pas vers l'optimum.

```
************************************ RESULTATS ****************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[4]) ;
            Ps : Suivis de leurs bases respectives :

32 12 19 5 dans la base : x5 x6 x7 x1

2 6 1 5 dans la base : x5 x3 x7 x1

1 7 2 3 dans la base : x5 x3 x8 x1


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

0 0 0 5 dans la base : x5 x6 x7 x1

0 6 0 5 dans la base : x5 x3 x7 x1

0 7 0 3 dans la base : x5 x3 x8 x1


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

5 0 0 0 dans la base : x1 x5 x6 x7

5 6 0 0 dans la base : x1 x3 x5 x7

3 7 0 0 dans la base : x1 x3 x5 x8
On ecrit les vecteurs dans la base canonique :

5 0 0 0 dans la base : x1 x2 x3 x4

5 0 6 0 dans la base : x1 x2 x3 x4

3 0 7 0 dans la base : x1 x2 x3 x4


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[4] (from First to Last) :

5 0 0 0

5 0 6 0

3 0 7 0

************************************ TERMINE ******************************************

userdz@ubuntu:~/U_Directory/TER$
```

À la convergence, on a :
$(x_1, x_2, x_3, x_4) = (3, 0, 7, 0)$.
D'où : **Z = 99**.

8.3.3 Suite exemple : Résolution par la 2ème Approche

Ci-dessous, les résultats de la 2ème approche. On converge cette fois vers l'optimum.

```
********************************** RESULTATS ****************************************

Ci-dessous les vecteurs des solutions intermediaires (dans |R[4]) ;
            Ps : Suivis de leurs bases respectives :

6 5 6 5 dans la base : x4 x6 x7 x8

1 1 7 5 dans la base : x4 x6 x3 x8

0.5 3 6.5 2 dans la base : x4 x1 x3 x8


Suite au Matchage Vect-Base on obtient les resultats ci-dessous :

6 0 0 0 dans la base : x4 x6 x7 x8

1 0 7 0 dans la base : x4 x6 x3 x8

0.5 3 6.5 0 dans la base : x4 x1 x3 x8


Apres mise en ordre croissant des bases et vecteurs on obtient les resultats suivants :

6 0 0 0 dans la base : x4 x6 x7 x8

7 1 0 0 dans la base : x3 x4 x6 x8

3 6.5 0.5 0 dans la base : x1 x3 x4 x8
On ecrit les vecteurs dans la base canonique :

0 0 0 6 dans la base : x1 x2 x3 x4

0 0 7 1 dans la base : x1 x2 x3 x4

3 0 6.5 0.5 dans la base : x1 x2 x3 x4


On redimensionne les VectSolu pr obtenir dans l'ordre les sommets a parcourir ci-apres :
        Ps : Ecrits dans la base canonique de |R[4] (from First to Last) :

0 0 0 6

0 0 7 1

3 0 6.5 0.5 <-----

********************************** TERMINE ******************************************

userdz@ubuntu:~/U_Directory/TER$
```

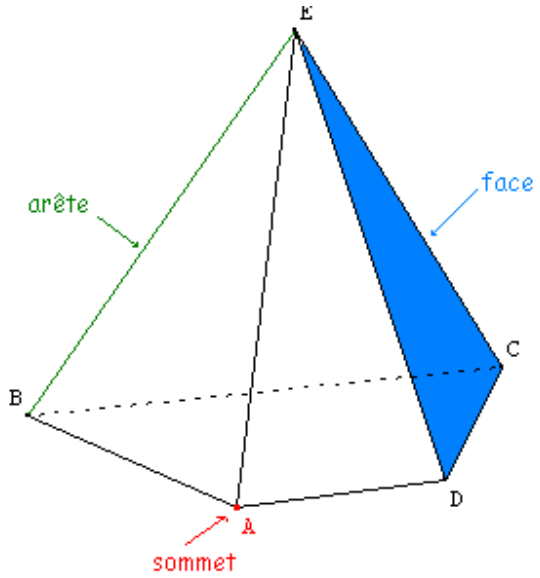
À la convergence, on a :
$(x_1, x_2, x_3, x_4) = (3, 0, 6.5, 0.5)$.
D'où : **Z = 100.5**.

8.4 Table des figures

8.5 Glossaire

8.5.1 Les polyèdres

Un polyèdre est un solide de l'espace délimité par un nombre fini de polygones, tel que chaque côté de chaque polygone est commun avec un côté d'un autre polygone. Les sommets des polygones sont appelés sommets du polyèdre, les côtés des polygones sont appelés arêtes du polyèdre, tandis que les polygones sont les faces du polyèdre.



Un polyèdre est dit convexe si, pour chaque plan de l'espace qui contient une face du polyèdre, le polyèdre est tout entier dans un des demi-espaces délimité par le plan. Pour ces polyèdres convexes, on a la célèbre relation d'Euler, qui relie le nombre de faces F, le nombre de sommets S, le nombre d'arêtes A :

$$F+S=A+2$$

8.5.2 Polyèdre réguliers

Il n'existe que 5 polyèdres convexes qui sont réguliers (c'est-à-dire que leurs faces sont des polygones réguliers). On les appelle aussi les solides platoniciens. Ils sont :

- Le tétraèdre régulier : 4 faces (des triangles équilatéraux), 4 sommets, 6 arêtes.
- Le cube : 6 faces (des carrés), 8 sommets et 12 arêtes.
- L'octaèdre régulier : huit faces (des triangles équilatéraux), 6 sommets, 12 arêtes.
- Le dodécaèdre régulier : 12 faces (des pentagones réguliers), 20 sommets, 30 arêtes.
- L'icosaèdre régulier : 20 faces (des triangles équilatéraux), douze sommets, et trente arêtes.

Remarquons que sur tous ces polyèdres, la relation d'Euler est vérifiée.

8.6 Bibliographie

- Simplexe :

  ⇨ Méthode d'optimisation combinatoire : I.CHARONE / A.GERMA / O.HUDRY
  Broché: 268 pages
  Editeur : Masson (1 décembre 1997)
  Collection : Collection pédagogique de télécommunication
  Langue : Français
  ISBN-10: 222585307X
  ISBN-13: 978-2225853074

- Qt :

  ⇨ http://www.qtsoftware.com/support-services/support
  ⇨ http://www.btsinfogap.org/cours/s3/ig2/cppqt/installqt.html

- OpenGL :

  ⇨ http://users.polytech.unice.fr/~buffa/cours/synthese_image/index.html
  ⇨ http://cat.inist.fr/?aModele=afficheN&cpsidt=7715797
  ⇨ http://www.geothalg.ulg.ac.be/cours1C/node165.html