Guide d'utilisation : Algorithme du Simplexe sous Scilab

Justine SAUCE

Objectif. Le but de ce court guide est d'expliquer plus en détail l'algorithme implémenté ainsi que son utilisation. Une partie sera dédiée à la compréhension du code et l'autre partie dédiée à l'utilisation de l'algorithme et plus particulièrement à la boîte de dialogue. Ce guide contient l'essentiel du code Scilab afin d'offrir un support plus facile à lire. Deux fichiers Scilab compose ce travail, le fichier de fonctions `Algorithme du Simplexe.sci` et le fichier d'instructions `Algorithme du Simplexe.sce`.

Problème. Le problème considéré est de minimiser la fonction objectif c, définie par,

$$z = c^T x \tag{1}$$

où, $c = (c_1, c_2, \ldots, c_n) \in \mathbb{R}^n$.

sous les p contraintes suivantes,

$$Ax = b, \quad x \geq 0 \tag{2}$$

avec,

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\ \vdots & \vdots & & \vdots \\ a_{p,1} & a_{p,2} & \ldots & a_{p,n} \end{pmatrix} \in \mathcal{M}_{p,n}(\mathbb{R}), \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix} \in \mathbb{R}^p$$

La forme de ce problème est appelée **forme standard**. Un problème d'optimisation classique que peut rencontrer une entreprise quand elle souhaite maximiser un bénéfice, par exemple, contient des contraintes d'inégalités, qui peuvent être liée à une disponibilité quantifiée de produit, une limite minimale ou maximale de production. Ces contraintes ne sont donc pas dans une forme standard. Le but de l'algorithme est donc de créer des variables d'écarts adaptées, et en nombre suffisant, pour optimiser un problème quelconque. Il devra prendre en considération autant la minimisation que la maximisation et se basera sur **la méthode du simplexe en deux phases**.

Table des matières

1 Construction de l'algorithme **2**
1.1 Pivot de Gauss 2
1.2 Indice de la variable sortante 3
1.3 Simplexe 4
1.4 Algorithme du Simplexe 9

2 Utilisation de l'Algorithme **10**
2.1 Exemple 1 : Les ceintures 10
2.2 Exemple 2 : Inégalités supérieures 11
2.3 Exemple 3 : Inégalités mélangées 12
2.4 Boîte de dialogue 13

1 Construction de l'algorithme

Pour l'implémentation de l'algorithme, nous utiliserons la méthode en deux phases. En effet, dans le cas de contraintes d'inégalité de type inférieur, les variables d'écart créées pour se ramener à une forme standard peuvent être choisies comme variable de base. Cependant, ce n'est pas le cas lorsqu'on rencontre des contraintes de type supérieur, il convient donc de mettre en oeuvre une première phase de recherche d'un sommet, qui constituera nos p variables de base initiales.

Remarques :

▷ On souhaite garder un œil sur la progression de l'algorithme, c'est pourquoi à chaque étape on choisit d'afficher la progression, dans le cas de problèmes en grande dimension, il est tout à fait possible de supprimer les commandes d'affichage.

▷ L'algorithme stocke les données de contraintes à l'aide du mode `sparse()`, ce qui optimisera le calcul dans le cas de problèmes de grande dimension.

▷ Il se compose de trois fonctions, `Pivot Gauss Jordan()`, `Indice variable sortante()` et `Simplexe()`. Chacune de ses fonctions est accompagnée de commentaires détaillant les étapes du code, on reviendra exclusivement sur les points clés.

▷ Des explications précéderont à chaque fois l'extrait de code donné.

1.1 Pivot de Gauss

La fonction `Pivot Gauss Jordan()` permet de réaliser le pivot nécessaire à chaque étape de la méthode du Simplexe, une fois que la variable entrante et la variable sortante sont choisies, on obtient notre pivot. Le pivot est réduit à 1 et les autres lignes sont réduites à 0 pour la colonne de référence du pivot. Cette fonction met aussi à jour l'état des contraintes et l'état de l'objectif. Le vecteur `L` fait référence à la ligne de coût (dernière ligne d'un tableau simplexe).

```
function [A,b,z,L]=Pivot_Gauss_Jordan(A,b,e,s,z,L)
    // Principe : Pivot de Gauss de la matrice A.
    //            Choix du pivot avec les variables entrantes et sortantes
    //            Mise à jour des contraintes du vecteur b
    //            Mise à jour du cout du vecteur L
    //            Mise à jour de l'objectif z

    // Entrees :
    //      A : matrice dont on souhaite effectuer le pivot
    //      b : vecteur des contraintes à mettre à jour
    //      e : indice variable entrante
    //      s : indice variable sortante
    //      z : objectif à mettre à jour
    //      L : couts à mettre à jour

    // Sorties :
    //      A : matrice mise à jour
    //      b : vecteur des contraintes mis à jour
    //      z : objectif mis à jour
    //      L : couts mis à jour
```

```
    // Nombre de contraintes p
    p = length(b);

    for i = 1:p
        // A est une matrice sparse : pour eviter des problèmes lors
        // de la division on definit des variables temporaires
        As = full(A(s,e));
        Ai = full(A(i,e));

        if (As<>0)
            // MaJ ligne pivot
            if(i == s) then
                b(i) = b(i)/As;
                A(i,:) = A(i,:)/As;
            // MaJ autres lignes
            else
                b(i) = b(i) - (Ai/As)*b(s);
                A(i,:) = A(i,:) - (Ai/As)*A(s,:);
            end;
        end;
    end;
    // MaJ objectif et couts
    z = z - (L(e))*b(s);
    L = L - (L(e))*A(s,:);
endfunction;
```

1.2 Indice de la variable sortante

Le choix de la variable sortante se fait une fois la variable entrante repérée, les ratios des contraintes sont calculés et le plus petit ratio non nul (et non négatif) correspondra à la variable sortante.

```
function s=Indice_variable_sortante(A,b,e)
    // Principe : Renvois l'indice de la variable sortant de la base

    // Entrees :
    //      A : matrice correpondant au tableau de simplexe courant
    //      b : vecteur des contraintes courant
    //      e : indice variable entrante

    // Sortie :
    //      s : indice variable sortante

    // Nombre de contraintes
    p = length(b);

    // On calcule les ratios
    for i=1:p,
        if A(i,e)>0 then
            R(i)=b(i)/A(i,e);
        else R(i)=%inf;
        end
    end
    // On recupère l'indice du plus petit ratio
    [val,s]=min(R);
endfunction;
```

1.3 Simplexe

Le code de la fonction étant un peu long, nous allons le découper en plusieurs parties afin de le commenter au fur et à mesure. La fonction `Simplexe()` correspond à la méthode des deux phases du simplexe pour la minimisation d'un problème.

Définition et paramètres.

```
function [xsol, z]=Simplexe(A,b,c,E)
    // Principe : Résolution du problème d'optimisation min z = <c,x> sous
    // les contraintes Ax = b

    // Entrees :
    //      A : matrice dont on souhaite effectuer le pivot
    //      b : vecteur des contraintes à mettre à jour
    //      c : vecteur de la fonction de cout
    //      E : vecteur des contraintes (1 : pour <= ou = / -1 : pour >=)

    // Sorties :
    //      xsol : solution optimale
    //         z : valeur optimale
```

Forme Standard.

La première étape de cette fonction est la création des variables d'écarts et des variables artificielles. La matrice de contraintes A sera transformée pour la Phase I et la Phase II. Avant d'entrer dans la première phase, A sera mise sous forme standard via la matrice `stdA`, pour cela nous introduisons des variables d'écarts, le coefficient sera 1 pour une contrainte d'égalité ou une contrainte d'inégalité de type inférieur et -1 pour un type supérieur.

```
    // Dimension du problème
    [p,n] = size(A);

    // Formulation du cout
    c = [c,zeros(1,p)]

    // Initialisation de l'optimum à 0
    z = 0;

    // Nouvelle forme de la matrice A : Passage en mode Matrice creuse
    // & ajout des variables d'écarts
    stdA = sparse([A,speye(p,p)]);
    for i=1:p
        stdA(i,n+i)=stdA(i,n+i)*E(i)
    end;
```

Initialisation Phase I.

Une fois la problème mis sous forme standard, on rentre dans la première phase, on commence par créer une matrice `tildeA` qui contiendra les coefficients des contraintes renseignées dans la matrice A mais en plus les variables d'écarts et autant de variables artificielles que de contraintes d'inégalités supérieures rencontrées.

Un autre point clé de la première phase est la composition et l'ordre des variables de base. Nous devons créer un vecteur contenant les variables de base, mais également dans le bon ordre afin de pouvoir facilement localiser la variable sortante. Dans la première phase, nous devons nous assurer que l'ordre des variables de base est respecté dans le cas d'un mélange entre des contraintes de type supérieur et inférieur.

Enfin, il faudra effectuer les premières opérations consistant à remettre à 0 les colonnes de la ligne de coût pour les variables artificielles.

```
// Nombre de variables artificielles a introduire
m = 0;
for i = 1:p
    if E(i) < 0
       m = m +1;
    end;
end;
// Nouvelle matrice A : ajout des variables artificielles
Vartif = [];
for i=1:p
    if E(i) < 0 then
        tamp = zeros(p,1);
        tamp(i)=1;
        Vartif = [Vartif,tamp];
    end;
end;
tildeA = sparse([stdA,Vartif]);
// Vecteur d'indices des variables de base
W = zeros(1,p);
for i=1:p+m
    j = find(tildeA(:,n+i)==1);
    W(j) = n+i;
end;
// Ligne du cout pour la première phase
L = [zeros(1,n+p),ones(1,m)];

// Operations d'initialisation
for i = 1:p
    for j = 1:m
        z = z - b(i)*tildeA(i,n+p+j);
        L = L - tildeA(i,:)*tildeA(i,n+p+j);
    end;
end;
// Point de départ
disp('Point de depart');
disp('Tableau du simplexe : ',full(tildeA));
disp('Cout : ',L);
disp('Contraintes : ',b);
disp('Objectif : ',z);
```

Phase I.

On va entrer dans la phase I. Pour cela on considère une boucle "tant que", qui effectuera la recherche de la variable entrante et sortante ainsi que la mise à jour des éléments du tableau du simplexe tant que la solution optimale n'est pas atteinte. Tant que la ligne de coût contient des coefficients négatifs, l'optimum de la Phase I n'est pas atteint. On sortira de la boucle lorsque le minimum de la ligne de coût sera supérieur à -1^{-10} pour éviter les problèmes liés à la précision des calculs. Enfin, on mettra à jour le vecteur contenant les variables de base avec la nouvelle variable entrante à la place de la variable sortante.

```
// Indice d'incrémentation
i = 0;

while(1)

    // On determine la variable entrante
    if (min(L)>=-1D-10)
        disp('Phase I terminee');
        break;
    end;

    // cout_minimal : valeur associee au cout minimal
    // e : indice associe au cout minimal
    [cout_minimal,e] = min(L);
    disp('Indice de la variable entrante : ',e);

    // Puis la variable de base sortante
    s = Indice_variable_sortante(tildeA,b,e);
    disp('Indice de la variable sortante : ',W(s));

    // On calcule alors le nouveau tableau à l'aide du Pivot de Gauss Jordan
    [tildeA,b,z,L]=Pivot_Gauss_Jordan(tildeA,b,e,s,z,L);

    // Mise à jour
    disp (string(i+1)+'e iteration : ');
    i = i+1;
    W(s)=e;

    // Progression
    disp ('Indice des variables de base :',W);
    disp('Tableau du simplexe : ',full(tildeA));
    disp('Cout : ',L);
    disp('Contraintes : ',b);
    disp('Objectif : ',z);

end;
```

Initialisation Phase II.

Une fois la phase I achevée, il faut revenir à un problème sans variables artificielles, pour cela on ne récupère **que** les informations obtenues via la phase I pour les variables du problème et les variables d'écart.

Cette fois-ci, on part de la fonction objectif pour notre ligne de coût, les variables d'écart auront des coefficients nuls pour la ligne de coût. On reprend donc le vecteur `c` construit lors de l'étape **Forme Standard.**

Sur le même principe que la Phase I, on réalise les opérations nécessaires pour remettre à 0 les colonnes possibles de la ligne de coût.

```
// On retire maintenant les variables artificielles
// Nouveau vecteur d'indices des variables de base
B = W(find(W<=(n+p)));

// Nouvelle forme standard de A
stdA = tildeA(:,1:(n+p));

// Indice d'incrémentation remis à 0
i = 0;

disp ('Phase II');

// Point de départ
    disp ('Indice des variables de base :',B);

    disp('Tableau du simplexe : ',full(stdA));
    disp('Cout : ',c);
    disp('Contraintes : ',b);
    disp('Objectif : ',z);

// Operations d'initialisation (pour les pivots à 1 on remet a 0 le cout)
for j=1:n+p
    i = find(B==j);
    if (stdA(i,j)==1) then
        z = z - c(j)*b(i);
        c = c - c(j)*stdA(i,:);
    end;
end;

// Point de départ après mise à jour
disp('Mise a jour du cout et de z');
disp('Cout',c);
disp('z',z);
```

Phase II.

Enfin, la Phase II est construite sur une boucle "tant que", comme pour la Phase I. Les étapes sont similaires. On affiche toujours la progression et cette fois-ci en sortant de la boucle on obtiendra la solution optimale ainsi que l'optimum, pour cela il suffit de réordonner l'état des contraintes en fonction des variables de base. Les variables hors base seront mises à 0.

```
    while(1)

        // Puis on determine la variable entrante
        // On commence par verifier qu'il y a encore un cout negatif
        if (min(c)>=-1D-10)
            disp('Solution optimale atteinte');
            break;
        end;

        // On recupere l'indice du minimum
        //      cout_minimal : valeur associee au cout minimal
        //                 e : indice associee au cout minimal
        [cout_minimal,e] = min(c);

        disp('Variable entrante :'+string(e));

        // On determine ensuite la variable de base sortante
        s = Indice_variable_sortante(stdA,b,e);
        disp('Variable sortante :'+string(B(s)));

        // On calcule alors le nouveau système à l'aide du Pivot de Gauss Jordan
        [stdA,b,z,c]=Pivot_Gauss_Jordan(stdA,b,e,s,z,c);

        // Mise à jour du vecteur des variables de base
        B(s)=e;

        // Incrementation
        i = i+1;

        // Progression de la methode
        disp (string(i+1)+'e iteration : ');
        disp ('Indice des variables de base :',B);
        disp('Cout',c);
        disp('A',full(stdA));
        disp('b',b);
        disp('z',z);
    end;

    // On recupère la solution en replacant les contraintes par rapport aux
    // variables de base.
    x = [zeros(n+p,1)];
    for i=1:p
        x(B(i)) = b(i);
    end;
    xsol = x(1:n,1);
endfunction;
```

Remarque. On choisit de renvoyer la solution optimale uniquement pour les variables du problèmes.

1.4 Algorithme du Simplexe

On a maintenant tous les outils nécessaires à l'utilisation de la méthode du simplexe. Pour pouvoir prendre un compte le type de problème, on rajoute une variable `pb` qui vaudra 1 dans le cas d'une maximisation et 2 pour une minimisation. L'appel de la fonction `Algorithme du Simplexe()` affichera les tableaux du simplexe mis à jour à chaque itération, ainsi que la solution optimale et l'optimum.

Pour éviter tout problème lié à la négativité de certaines contraintes, une transformation des contraintes négatives en contraintes positives est également effectuée.

```
function [x,z]= Algorithme_du_Simplexe(A,b,c,pb,E)
    // Principe : Résolution du problème d'optimisation min/max z = <c,x> sous
    // les contraintes Ax = b en utilisant la méthode du simplexe (2 phases).

    // Entrees :
    //       A : matrice dont on souhaite effectuer le pivot
    //       b : vecteur des contraintes à mettre à jour
    //       c : vecteur de la fonction de cout
    //      pb : type de problème (1 : Maximisation / 2 : Minimisation)
    //       E : vecteur des contraintes (1 : pour <= ou = / -1 : pour >=)

    // Sorties :
    //      x : solution optimale
    //      z : valeur optimale

    // Dimension du probleme
    [p,n] = size(A);

    // Correction pour les contraintes négatives
    for i=1:p
        if b(i)<0 then
            A(i,:) = -A(i,:)
            b(i) = -b(i)
        end;
    end;

    // Cas de la Maximisation
    if pb==1 then
        [x, z]= Simplexe(A,b,-c,E);
        disp ('La solution optimale est ',x);
        disp ('L optimum est ', z);
    end;

    // Cas de la Minimisation
    if pb==2 then
        [x, z]= Simplexe(A,b,c,E);
        disp ('La solution optimale est ',x);
        disp ('L optimum est ', -z);
    end;

endfunction
```

2 Utilisation de l'Algorithme

L'algorithme est accompagné de plusieurs exemples regroupant différents cas afin de vérifier son bon fonctionnement. Dans cette partie on donnera le problème associé à chacun de ses exemples et les résultats obtenus concernant la solution optimale et l'optimum. Ces exemples ont pour but d'illustrer comment on passe d'un problème donné à notre liste de paramètres.

La dernière partie du fichier d'exécution scilab est accompagnée d'une boîte de dialogue, permettant à l'utilisateur de définir, plus aisément, les paramètres du problème de son choix.

2.1 Exemple 1 : Les ceintures

Problème. On considère le problème suivant.

$$\langle \text{E1} \rangle : \max z = 2x_1 + \frac{2}{3}x_2, \qquad \begin{aligned} 2x_1 + x_2 &\leq 1000 \\ x_1 + x_2 &\leq 800 \\ x_1 &\leq 400 \\ x_2 &\leq 700 \\ x_1 \geq 0, x_2 &\geq 0 \end{aligned}$$

Ici, il s'agit d'une maximisation, la variable `pb` vaudra 1. le problème est en dimension $n = 2$ et il y a $p = 4$ contraintes à prendre en compte.

La fonction objectif `c`, la matrice de contraintes `A`, le vecteur du type de contraintes `E` et le vecteur de contraintes `b` sont définis comme suit.

$$c = \begin{pmatrix} 2 & \frac{2}{3} \end{pmatrix} \in \mathbb{R}^2, \quad A = \begin{pmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \in \mathcal{M}_{4,2}(\mathbb{R}), \quad E = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \in \mathbb{R}^4, \quad b = \begin{pmatrix} 1000 \\ 800 \\ 400 \\ 700 \end{pmatrix} \in \mathbb{R}^4.$$

Le problème sera donc codé de la façon à l'aide des paramètres suivants.

```
A = [2,1;1,1;1,0;0,1];
b = [1000,800,400,700]';
c = [2,3/2];
E = [1,1,1,1]';
pb = 1;

Algorithme_du_Simplexe(A,b,c,pb,E);
```

On obtient les résultats ci-dessous.

```
"La solution optimale est "

     200.
     600.

"L optimum est "

     1300.
```

Pour donner un peu de contexte, z correspond à un bénéfice, x_1 à une quantité de ceinture de Type I produites et x_2 la quantité de ceinture de Type II. Les contraintes reflètent les conditions relatives à la capacité de production et l'objectif se concentre sur les coûts de production. Ainsi, le résultat indique que pour maximiser le bénéfice, il faut produire 200 ceintures de Type I et 600 ceintures de Type II. Le bénéfice maximal sera de 1300 euros.

2.2 Exemple 2 : Inégalités supérieures

Problème. On considère le problème suivant.

$$\langle \text{E2} \rangle : \min z = x_1 + 3x_2, \qquad \begin{aligned} -3x_1 - 2x_2 &\geq -60 \\ x_1 + x_2 &\geq 60 \\ x_2 &\geq 20 \\ x_1 \geq 0, x_2 &\geq 0 \end{aligned}$$

On s'intéresse à une minimisation, la variable `pb` vaudra 2. le problème est en dimension $n = 2$ et il y a $p = 3$ contraintes à prendre en compte.

La fonction objectif `c`, la matrice de contraintes `A`, le vecteur du type de contraintes `E` et le vecteur de contraintes `b` sont définis comme suit.

$$c = \begin{pmatrix} 1 & 3 \end{pmatrix} \in \mathbb{R}^2, \quad A = \begin{pmatrix} -3 & -2 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \in \mathcal{M}_{3,2}(\mathbb{R}), \quad E = \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \in \mathbb{R}^3, \quad b = \begin{pmatrix} -60 \\ 60 \\ 20 \end{pmatrix} \in \mathbb{R}^3.$$

```
A = [-3,-2;1,1;0,1];
b = [-60,60,20]';
c = [1,3];
E = [-1,-1,-1]';
pb = 2;

Algorithme_du_Simplexe(A,b,c,pb,E);
```

Cela nous donne les résultats suivants.

```
"La solution optimale est "

 40.000000
 20.

"L optimum est "

 100.00000
```

2.3 Exemple 3 : Inégalités mélangées

Problème. On considère le problème suivant.

$$\langle \text{E3} \rangle : \max z = 4x_1 + 3x_2, \qquad \begin{aligned} 5x_1 + 4x_2 &\leq 5000 \\ 8x_1 + 5x_2 &\leq 7300 \\ 2x_1 + 3x_2 &\geq 2700 \\ x_1 \geq 0, x_2 &\geq 0 \end{aligned}$$

On s'intéresse à une maximisation, la variable `pb` vaudra 1. le problème est en dimension $n = 2$ et il y a $p = 3$ contraintes à prendre en compte.

La fonction objectif `c`, la matrice de contraintes `A`, le vecteur du type de contraintes `E` et le vecteur de contraintes `b` sont définis comme suit.

$$c = \begin{pmatrix} 4 & 3 \end{pmatrix} \in \mathbb{R}^2, \quad A = \begin{pmatrix} 5 & 4 \\ 8 & 5 \\ 2 & 3 \end{pmatrix} \in \mathcal{M}_{3,2}(\mathbb{R}), \quad E = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \in \mathbb{R}^3, \quad b = \begin{pmatrix} 5000 \\ 7300 \\ 2700 \end{pmatrix} \in \mathbb{R}^3.$$

```
A = [5,4;8,5;2,3];
b = [5000,7300,2700]';
c = [4,3];
E = [1,1,-1]';
pb = 1

Algorithme_du_Simplexe(A,b,c,pb,E);
```

Les résultats suivants sont obtenus.

```
"La solution optimale est "

 600.
 500.

"L optimum est "

 3900.
```

2.4 Boîte de dialogue

Le dernier point à aborder est l'utilisation de la boîte de dialogue. Cette boîte de dialogue respecte exactement les formats des paramètres expliqués ci-dessus. Elle vise à faciliter le codage des paramètres pour les utilisateurs qui ne sont pas familiers avec Scilab.

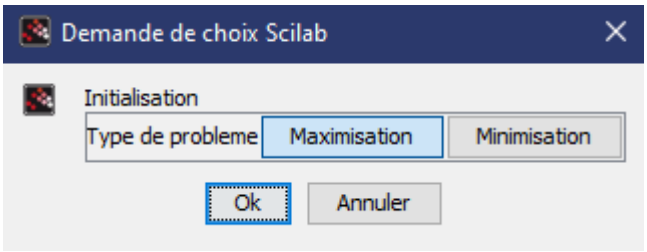
Le code utilise des fonctions déjà implémentées dans l'interface graphique de Scilab et il sera donné en fin de section.

Lorsque vous lancez le programme associé à la boîte de dialogue, une série de fenêtres contextuelles s'ouvrent pour vous demander d'entrer les paramètres du problème. Pour illustrer son utilisation on reprend l'**Exemple 3.**

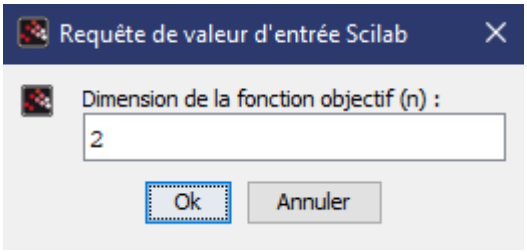
$$\langle \text{E3} \rangle : \max z = 4x_1 + 3x_2, \qquad \begin{aligned} 5x_1 + 4x_2 &\leq 5000 \\ 8x_1 + 5x_2 &\leq 7300 \\ 2x_1 + 3x_2 &\geq 2700 \\ x_1 \geq 0, x_2 &\geq 0 \end{aligned}$$

Les fenêtres sont présentées dans l'ordre et déjà remplies dans le cadre du problème.

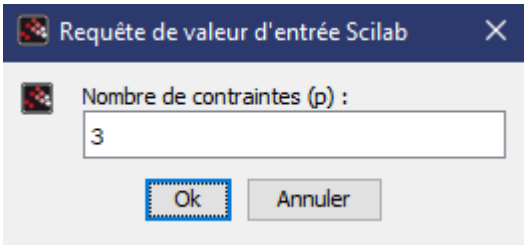
Spécification du type de problème.



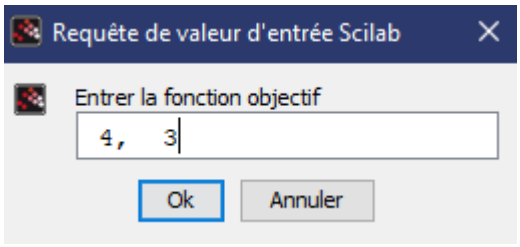
Dimension du problème.



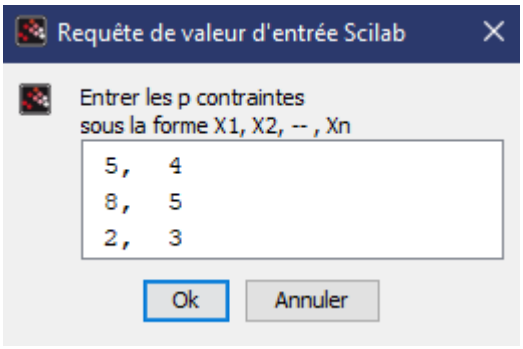
Nombre de contraintes.



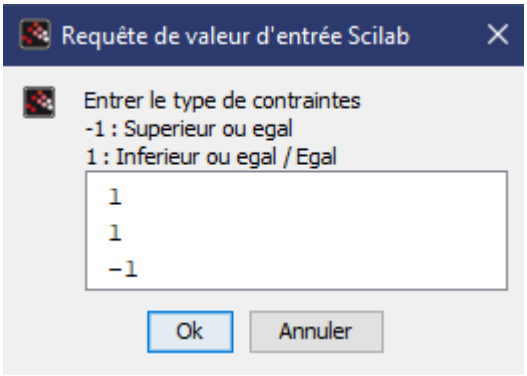
Déclaration de la fonction objectif.



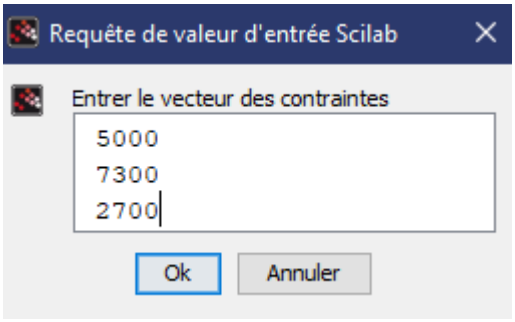
Déclaration de la matrice de contraintes.



Déclaration du type de contraintes.



Déclaration du vecteur des seuils de contraintes.



L'algorithme se lance dès que tous les paramètres ont été spécifiés. En cas d'erreur, annuler et redémarrer le programme.

Code de la boîte de dialogue.

```
l=list('Type de probleme',1,['Maximisation', 'Minimisation']);
pb=x_choices('Initialisation',list(l));

n=evstr(x_dialog('Dimension de la fonction objectif (n) :','2'))
p=evstr(x_dialog('Nombre de contraintes (p) :','3'))

c=evstr(x_matrix('Entrer la fonction objectif',zeros(1,n)))
A=evstr(x_matrix(['Entrer les p contraintes';'sous la forme X1, X2, -- , Xn'],
                 zeros(p,n)))
E=evstr(x_matrix(['Entrer le type de contraintes';'-1 : Superieur ou egal';'
                 1 : Inferieur ou egal / Egal'], ones(p,1)))
b=evstr(x_matrix('Entrer le vecteur des contraintes',zeros(p,1)))

Algorithme_du_Simplexe(A,b,c,pb,E);
```