TP 4 : Méthode du simplexe.

1 Le problème

On s'intéresse ici à la résolution d'un problème de programmation linéaire de la forme

$$\min\ C^T x,\ \text{tel que}\ \begin{cases} Ax = b, \\ x \geq 0, \end{cases} \tag{1}$$

où C est un vecteur de $\mathbb{R}^n$, A une matrice de taille $m \times n$ de rang m avec $m \leq n$, et b un vecteur de $\mathbb{R}^m$. On a adopté la notation $\mathbb{R}^n \ni v \geq 0$ si et seulement si $v_i \geq 0$ pour $i = 1, \ldots, n$. Le problème (1) est la forme dite standard de problème de programmation linéaire. On se propose de le résoudre par la *méthode du simplexe*. Dans la pratique les problèmes de programmation linéaire sont donnés sous la forme

$$\min\ C^T x,\ \text{tel que,}\ \begin{cases} Ax \leq b, \\ x \geq 0, \end{cases} \tag{2}$$

où $b \geq 0$, ce qui a un certain avantage lors de l'initialisation de l'algorithme du simplexe.
En guise d'application, on pourra considérer des problèmes modèles à sa convenance. Néanmoins nous fournissons ici quelques exemples simples :

Exemple 1 : solution attendue : $x_1 = 4, x_2 = 5, x_3 = 0$

$$\min\ x_1 - 3x_2 + 2x_3,\ \ \text{tel que,}\ \begin{cases} 3x_1 - x_2 + 2x_3 \leq 7, \\ -2x_1 + 4x_2 \leq 12, \\ -4x_1 + 3x_2 + 8x_3 \leq 10, \\ x \geq 0. \end{cases} \tag{3}$$

Exemple 2 : problème dû à Klee, Minty et Chvatal .
Pour $n \in N^*$ on considère le problème

$$\max \sum_{j=1}^{n} 10^{j-1} x_j,\ \text{tel que,}\ \begin{cases} x_i + 2\displaystyle\sum_{j=i+1}^{n} 10^{j-i} x_j \leq 10^{2n-2i},\ i = 1, 2 \ldots, n, \\ x \geq 0. \end{cases} \tag{4}$$

On peut montrer que pour ce problème, le domaine admissible est formé de 2^n sommets. Par ailleurs la méthode du simplexe initialisée comme décrit ci-dessous (c'est-à-dire les variables d'écart forment la base initiale), visite tous les 2^n sommets avant de fournir la solution. On prendra $n = 6$.

Exemple 3 : solution attendue : $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0$

$$\min\ -\frac{3}{4}x_1 + 20x_2 - \frac{1}{2}x_3 + 6x_4,\ \text{tel que}\ \begin{cases} \frac{1}{4}x_1 - 8x_2 - x_3 + 9x_4 \leq 0, \\ \frac{1}{2}x_1 - 12x_2 - \frac{1}{2}x_3 + 3x_4 \leq 0, \\ x_3 \leq 1, \\ x \geq 0. \end{cases} \tag{5}$$

Pour ce problème un **cyclage** est apparent. Valeur minimale attendue $-\frac{5}{4}$.

2 L' algorithme

La méthode du simplexe est un algorithme de résolution du problème (1). Elle peut être assimilée à une méthode itérative de type bloc-relaxation pour la résolution du système linéaire $Ax = b$, basée sur une décomposition de la matrice $A = [A_{\mathcal{B}}, A_{\mathcal{N}}]$, où $\{1, \ldots, n\} = \mathcal{B} \cup \mathcal{N}$ est une partition de l 'ensemble des indices de colonnes de A. La particularité ici est que l'ensemble $\mathcal{B}$ appelé *base* est mis à jour à chaque itération, par ajout et suppression d'un élément. Le critère de sélection des indices à ajouter et à supprimer vise simplement à garantir la décroissance de la fonctionnelle coût sous l'hypothèse que la solution devra être telle que $x_{\mathcal{N}} = 0$ et que $x_{\mathcal{B}} = A_{\mathcal{B}}^{-1} b \geq 0$.
L'algorithme peut être formulé comme une méthode de descente où à chaque itération, on détermine une direction de desctente et un pas optimal.

Algorithme du simplexe

1. **Initialisation.** (Ceci peut faire appel en pratique à un autre problème de programmation linéaire).
   Partitionner l'ensemble des indices $\{1, \ldots, n\} = \mathcal{B} \cup \mathcal{N}$, où $\mathcal{B}$ désigne la base et $\mathcal{N} = \{1, \ldots, n\} \setminus \mathcal{B}$.
   Attacher à ce partitionnement les partitionnements en colonnes $A_{\mathcal{B}}, A_{\mathcal{N}}$ de A, ainsi que les partitionnements $C_{\mathcal{B}}$, $C_{\mathcal{N}}$ de C et $x_{\mathcal{B}}$, $x_{\mathcal{N}}$ de x.
   La solution initiale x doit être telle que $x_{\mathcal{N}} = 0, x_{\mathcal{B}} = A_{\mathcal{B}}^{-1}\, b \geq 0$.
2. **Itérations.** Tant qu'on n'a pas convergé et que le problème n'est pas non borné :
   (a) Calculer le coût réduit : $S_{\mathcal{N}} = C_{\mathcal{N}} - A_{\mathcal{N}}^T A_{\mathcal{B}}^{-T} C_{\mathcal{B}}$
   (b) Analyser le coût réduit (ou déterminer l'indice (*entrant*) $q \in N$ i.e devant intégrer la base B) :
   — Si $S_{\mathcal{N}} \geq 0$ alors on a atteint l'optimum et l'algorithme **a convergé**.
   — Sinon calculer le nouvel indice rentrant dans la base :
   prendre $q \in \mathcal{N}$ tel que $S_q < 0$. Dans la pratique on prend
   $q = \arg\max_{j \in \mathcal{N}} \{|S_j|, j \in \mathcal{N}, S_j < 0\}$. On signale que c'est aussi ici que l'on peut contrôler le **cyclage**.
   (c) Chercher une nouvelle direction de descente (ou déterminer l'indice (sortant) p i.e devant quitter la base $\mathcal{B}$)
   — Calculer $d = A_{\mathcal{B}}^{-1} A_q$, où A_q est la colonne de A correspondant à l'indice (entrant) q obtenu ci dessus.
   — Si $d \leq 0$ alors le problème est **non borné** .
   — Sinon
   — déterminer l'indice $p = \arg\min_{i \in \mathcal{B}} \{\frac{(x_{\mathcal{B}})_i}{d_i}, d_i > 0\}$,
   — déterminer le pas optimal de descente $\rho = \frac{(x_{\mathcal{B}})_p}{d_p}$.
   (d) Mettre à jour les données
   — Solution : $x_{\mathcal{B}} = x_{\mathcal{B}} - \rho d, \quad x_{\mathcal{N}} = (0, \ldots, 0, \rho, 0, \ldots, 0) \equiv \rho e_q$, où e_q est le $q_{\mathcal{N}}$-ième vecteur de la base canonique de $R^{\mathrm{card}(\mathcal{N})}$, avec $q_{\mathcal{N}}$ l'indice de q dans $\mathcal{N}$.
   — Base : $\mathcal{B} = (\mathcal{B} \setminus \{p\}) \cup \{q\}, \qquad \mathcal{N} = \{1, \ldots, n\} \setminus \mathcal{B}$.
   — Solution : mettre à jour $x_{\mathcal{B}}$ et $x_{\mathcal{N}}$, car la base a changé. (*Etape non nécessaire si l'on met en oeuvre l'algorithme avec un vecteur x contenant $x_{\mathcal{B}}$ et $x_{\mathcal{N}}$*).

3 La mise en oeuvre

Afin de simplifier la mise en oeuvre et en particulier l'initialisation, on suppose le problème donné sous la forme (2). En effet sous cette forme on peut introduire les variables d'écart S (*slack varaibles en angfais*) (voir cours) et déduire directement une base initiale $\mathcal{B}$ du problème transformé, comme étant l'ensemble formé des indices des variables d'écart dans le problème transformé.

Puisque nous sommes sous `Matlab`, nous n'allons pas extraire explicitement les blocs $A_{\mathcal{B}}$, $A_{\mathcal{N}}$ etc. Il suffit en effet d'identifier les éléments de $\mathcal{B}$ par un tableau IB d'éléments de $\{1,\ldots,n\}$. Dès lors, on sait que $A_{\mathcal{B}} = A(:,\mathrm{IB})$. Ainsi, au terme de la résolution, la solution du problème (2) sera un vecteur de taille $n+m$ où m est le nombre des variables d'écart. Il faudra alors extraire judicieusement la solution du problème de la variable x !

Dans la suite, pour ne pas alourdir la description d'une fonction demandée, ses arguments d'entrée sont choisis de telle sorte que leur sens puisse s'obtenir en observant les arguments de sortie des fonctions qui la précèdent.

1. Fournir une fonction

```
function [co,Ao,bo,xo,IB,IN] = ajout_variables_ecart(c, A, b)
```

qui transforme le problème donné sous forme (2) en un problème sous forme standard (1), en ajoutant les variables d'écart. Elle prendra en arguments `(c, A, b)` les données (c, A, b) du problème (2) et retournera les données `(co, Ao, bo)` qui sont les données (c, A, b) du problème (1) ainsi que la solution initiale `xo` du problème (1) et les vecteurs `IB` et `IN` contenant respectivement les éléments de $\mathcal{B}$ et $\mathcal{N}$.

2. Fournir une fonction

```
function [cr] = cout_reduit(co, Ao, bo, xo, IB,IN)
```

qui calcule le coût réduit.

3. Fournir une fonction

```
function [est_optimal, jNSortant] = analyse_cout_reduit(cr)
```

qui analyse le coût réduit et retourne deux valeurs donc le premier indique si le problème a convergé et le second fournit l'indice entrant dans la base, noté q dans l'algorithme. L'indice de colonne de A entrant dans la base $\mathcal{B}$ étant sortant de $\mathcal{N}$; et comme le coût réduit est défini par rapport à $\mathcal{N}$, on convient de définir q relativement à `IN` (i.e `q = IN(jNsortant)`).

4. Fournir une fonction

```
function [dB,dN]
         = composantes_direction_descente(co, Ao, bo, xo, IB, IN, jNSortant)
```

qui calcule les composantes basiques (`dB`) et non basiques (`dN`) de la nouvelle direction de descente.

5. Fournir une fonction

```
function [pb_non_borne, iBsortant, pas_optimal]
          = analyse_directions_descente(co,Ao,bo,xo,IB,IN,dB,dN)
```

qui analyse les composantes (`dB`) et (`dN`) de la direction de descente et retourne des variables informant : si le problème est non borné (`pb_non_borne`), indiquant l'indice sortant de la base (`iBsortant`) noté p dans l'algorithme (ici aussi `iBsortant` est relatif à la base i.e. `p = IB(iBsortant)`) et fournissant le pas optimal dans la direction de descente (`pas_optimal`) noté ρ dans l'algorithme.
6. Fournir une fonction

```
function [newxo, newIB, newIN]
= nouvelle_solution(co,Ao,bo,xo,IB,IN,dB,dN,pas_optimal,iBsortant,jNsortant)
```

qui met à jour la solution et la base $\mathcal{B}$ ainsi que son complémentaire $\mathcal{N}$.
6. Fournir une fonction

```
function [x, iter] = methode_simplexe_min(c, A ,b, boucle_max)
```

qui met en oeuvre l'algorithme du simplexe pour déterminer la solution du problème de minimisation (2), en effectuant un maximum de `boucle_max` itérations. Elle retourne la solution du problème et le nombre d'itérations effectuées.

4 Validation

En considérant soit des problèmes de programmation linéaire de votre choix, soit les exemples fournis ci-dessus, créer un script `scriptTP4.m` qui applique l'algorithme du simplexe à la résolution d'un problème de programmation linéaire.