CHAPTER 1

TP 1 : Programmation linéaire

Un compte rendu rapide de ce TP sera à rendre pour le jeudi 20 février. Il contiendra, pour chaque exercice, une description rapide du modèle, avec les hypothèses faites, les commandes utilisées pour le résoudre, le résultat, et tout commentaire que vous jugerez utile.

Tous les exercices de cette fiche peuvent être faits avec MuPAD, Maple ou probablement la plupart des systèmes de calcul formel, ainsi qu'avec Matlab ou Scilab. Notez que MuPAD permet de résoudre des programmes linéaires en entiers, et il permettra, dans des séances futures, de manipuler des tableaux pour exécuter pas à pas l'algorithme du simplex (comme en cours !). Pour commencer sous MuPAD, le mieux est de suivre le tutorial que l'on lance avec :

```
?tutorial
```

Exemple 1.0.1. On considère le programme linéaire suivant :

Maximiser : $-x + y + 2z$

Sous les contraintes :

- $3x + 4y - 3z \leq 23$
- $5x - 4y - 3z \leq 10$
- $7x + 4y + 11z \leq 30$
- $x, y, z \geq 0$

Voici les commandes pour résoudre le problème sous MuPAD :

```
c :=[ { 3*x + 4*y - 3*z <= 23,
        5*x - 4*y - 3*z <= 10,
        7*x + 4*y + 11*z <= 30 },
        -x + y + 2*z,
        NonNegative
    ] :
linopt : :maximize(c);
```

Pour plus d'information, voir la documentation de la bibliothèque de programmation linéaire linopt avec :

```
?linopt
```

Sous `Maple`, vous pouvez utiliser la fonction `simplex[maximize]` qui est très similaire.

Exemple 1.0.2. Pour l'utilisation de `Scilab`, je recommande le site suivant (en francais !) :

`http://cermics.enpc.fr/scilab/`

Vous trouverez en particulier des exemples de résolution de programmes linéaires dans les TPs sur l'optimisation :

Questions :: `http://cermics.enpc.fr/scilab/Optimisation/Optim_Q_statique/`

Réponses :: `http://cermics.enpc.fr/scilab/Optimisation/Optim_R_statique/`

Exercice 1. Résoudre le problème du régime de Polly

Exercice 2. Un bûcheron a 100 hectares de bois de feuillus. Couper un hectare de bois et laisser la zone se régénérer naturellement coûte 10 kF par hectares, et rapporte 50 kF. Alternativement, couper un hectare de bois, et replanter avec des pins coûte 50 kF par hectares, et rapporte à terme 120 kF. Sachant que le bûcheron n'a que 4000 kF en caisse au début de l'opération, déterminer la meilleure stratégie à adopter et le profit escomptable.

EXERCICE 3. Vous êtes chargé de la planification d'un chantier qui nécessite le traitement de 9 tâches décrites dans le tableau suivant. Il peut être nécessaire, pour traiter une tâche donnée, (par ex. F) que le traitement d'autres tâches soit terminé (ici C et E). De plus, on peut choisir d'accélérer le traitement d'une tâche donnée, moyennant un surcoût par jours de réduction ; le nombre maximum de jours de réduction et le sûrcout dépendent de la tâche.

Tâches	Dépendences	Durée (jours)	Réd. max (jours)	Coût de réd (kEuro/jours)
A		2	1	9
B	A	7	2	2
C	D	4	3	1
D		2	0	
E	A	3	1	8
F	C,E	2	1	5
G	A,F	3	1	7
H	F,D	3	2	6
I	B,H	3	0	

Quelle est la durée minimale du chantier sans surcoût ?

Peut-on effectuer le chantier en moins de 8 jours ?

Quelle est le coût minimal pour finir le chantier en 10 jours ?

Quelle est la durée minimale du chantier avec un surcoût de 15 au plus ?

Contrainte supplémentaire : vous *devez* être paresseux quand il s'agit de taper un programme linéaire ; l'ordinateur l'est aussi pour le traiter. Donc, pour chacune de ces questions, vous essayerez de trouver le programme linéaire le plus petit possible.

EXERCICE 4. S'il vous reste du temps, profitez en pour essayer Scilab (par ex. regarder les autres outils d'optimisation disponibles) et MuPAD (par ex. commencer le tutorial)

CHAPTER 2

TP2 : Simplexe et algèbre linéaire

Les exercices de cette fiche peuvent être faits avec MuPAD, ou probablement la plupart des systèmes de calcul formel.

EXERCICE 5. Utilisez la bibliothèque `linopt`, et en particulier les fonctions pour manipuler les tableaux (`linopt : :Transparent`) pour vérifier pas à pas vos résolutions de programmes linéaires par l'algorithme du simplexe. Vous pouvez vous récupérer la plupart des programmes linéaires du Chvatal a l'adresse suivante `http://www.lapcs.univ-lyon1.fr/~nthiery/RO/Notes/tableaux.mu`.

Le but du reste de la séance est de vous familiariser avec le système de calcul formel MuPAD, en particulier pour programmer. La documentation de MuPAD contient un tutorial très complet et bien fait ; pour y accéder il suffit de taper ?tutorial. Il existe une version française du tutorial, mais je ne sais pas si elle est installée.
Pour vous guider, voici quelques exercices de programmation :

EXERCICE 6. Programmer une procédure (voir ?proc, ?DOM_LIST et ?for) qui prends une liste comme argument, et renvoie le plus grand élément de cette liste.

EXERCICE 7. Programmer une procédure qui prend une matrice M (voir ?matrix) et un coefficient c, et multiplie tous les coefficients de la matrice par c. Évidemment, le plus simple est d'écrire `c*M`, mais ce n'est pas l'objectif de l'exercice.

EXERCICE 8. Programmer une procédure qui prend une matrice M, et lui applique le pivot de Gauss.

CHAPTER 3

TP3 : Programmation et récursivité

Les exercices de cette fiche peuvent être faits avec MuPAD, Maple, ou n'importe quel langage de programmation.

Exercice 9. Programmer une procédure récursive qui prend un entier n, et renvoie le $n$ième nombre de Fibonnacci. Regarder la différence de temps d'exécution selon si l'on utilise l'option `remember`, ou pas (voir ?proc).

Exercice 10. Programmer une procédure récursive `listeDeBits` qui prend un entier n, et renvoie toutes les listes de 0 et de 1 de longueur n. Par exemple :

(1) `listeDeBits(0) = [[]]`
(2) `listeDeBits(1) = [[0],[1]]`
(3) `listeDeBits(2) = [[0,0],[0,1],[1,0],[1,1]]`

Exercice 11. Programmer une procédure récursive `compositions` qui prend un entier n, et renvoie toutes les listes d'entiers strictement positifs dont la somme vaut n (une telle liste est appelée *composition* de n). Par exemple :

(1) `compositions(0) = [[]]`
(2) `compositions(1) = [[1]]`
(3) `compositions(2) = [[2],[1,1]]`
(4) `compositions(3) = [[3],[2,1],[1,2],[1,1,1]]`

Exercice 12. Programmer une procédure récursive `partitions` qui prend un entier n et un entier k, et renvoie toutes les listes décroissantes d'entiers entre 1 et k dont la somme vaut n (une telle liste est appelée *partition* de n dont les parts sont bornées par k). Par exemple :

(1) `partitions(0,10) = [[]]`
(2) `partitions(1,10) = [[1]]`
(3) `partitions(2,10) = [[2],[1,1]]`
(4) `partitions(3,10) = [[3],[2,1],[1,1,1]]`
(5) `partitions(3,2) = [[2,1],[1,1,1]]`
(6) `partitions(4,10) = [[4],[3,1],[2,2],[2,1,1],[1,1,1,1]]`
(7) `partitions(4,2) = [[2,2],[2,1,1],[1,1,1,1]]`

Exercice 13. Problème du sac-à-dos : programmer une procédure récursive qui prend une liste de k entiers strictements positifs w et un entier n, et renvoie une liste l de k entiers positifs qui maximise la fonction objective

$$z := w[1] * l[1] + w[2] * l[2] + \cdots + w[k] * l[k]$$

sous la contrainte que z ne doit pas dépasser n. Par exemple :

(1) `sacados([1],2) = [2]`
(2) `sacados([1,1],2) = [2,0] (ou [1,1] ou [0,2])`
(3) `sacados([5,2,1],7) = [2,0,1] (ou [0,1,1])`
(4) `sacados([5,2],8) = [1,1]`

CHAPTER 4

TP 4 : Réseaux de transports

Les exercices de cette fiche peuvent être faits avec MuPAD, Maple, ou probablement la plupart des systèmes de calcul formel. La bibliothèque Network de MuPAD (networks de Maple) permet de manipuler des réseaux de transport.

Il y a quelques bugs dans la bibliothèque Network de MuPAD <= 2.5.2 qui sont corrigés dans le fichier suivant : `http://www.lapcs.univ-lyon1.fr/~nthiery/RO/Notes/networkfix.mu`. Il suffit de le télécharger, et de le lire au début de la session MuPAD avec la commande

```
read("<chemin>networkfix.mu") :
```

D'autre part, cette bibliothèque fonctionne mal en calculs approchés (le programmeur n'a pas été prudent avec les tests à zéro, d'où des instabilités numériques). Il est donc prudent de ne faire que des calculs avec des nombres rationnels plutôt que des flotants.

Exercice. Consulter la documentation de `Network : :minCost`, et essayer l'exemple proposé.

Vérifiez la solution optimale que l'on avait obtenu pour le premier exemple du cours sur les réseaux de transport.

Exercice. Résolution du problème d'assignement avec les réseaux de transport.

Modéliser le problème du cours de l'assignement cours/prof et le résoudre.

On suppose maintenant que l'utilisateur fournit une matrice décrivant un problème d'assignement similaire. Voici par exemple la matrice du problème du cours (on a transformé les nombres flotants en rationnels) :

```
matrix([[3, 4, 23/10, 29/10, 28/10],
        [225/100, 32/10, 37/10, 19/10, 44/10],
        [26/10, 37/10, 45/10, 27/10, 31/10],
        [39/10, 41/10, 26/10, 39/10, 24/10],
        [28/10, 28/10, 35/10, 34/10, 42/10]]);
```

Écrire une procédure qui prend un telle matrice comme argument, et renvoit l'assignement optimal. Note : ?for, ?proc, et ?linalg : :matdim sont vos amis !

Exercice. L'objectif est maintenant d'évaluer la complexité en moyenne de l'algorithme précédent, en fonction de la taille n de la matrice.

Pour chaque entier n entre 1 et 10 (ou beaucoup plus si vous pouvez), tirer un certain nombre de matrices aléatoires de taille $n \times n$, et faire la moyenne du temps de calcul (cf. ?random et ?time). Tracer la courbe obtenue (cf. ?plot), et essayer de trouver une fonction de la forme an^d qui ajuste cette courbe au mieux.

Essayer de majorer au mieux la complexité au pire théorique de cet algorithme.

Conclusion ?

Théorème. *(Théorème de König ou lemme des mariages) On a un ensemble de n filles et n garçons que l'on veut marier ensemble. Dans notre grande magnanimité, on veut bien faire attention à ne pas marier deux personnes qui ne se connaissent pas.*

Si chaque fille connait exactement $k \geq 1$ garçons et chaque garçon connaît exactement k filles, alors on peut arranger n mariages de façon à ne pas marier des inconnus.

Exercice. Prouvez ce théorème en construisant le problème de transport qui va bien.

CHAPITRE 5

TP 5 Ordres partiels et ordonnancement

Les exercices de cette fiche peuvent être faits avec MuPAD, Maple, ou probablement la plupart des systèmes de calcul formel. La bibliothèque Network de MuPAD (networks de Maple) permet de manipuler des graphes.

Le but de ce TP est d'implanter quelques algorithmes sur les ordres partiels, et en particulier la recherche par branchement et coupe d'un ordonnancement optimal UET d'un ordre partiel sur un nombre donné de processeurs.

On représentera un ordre partiel par son diagramme de Hasse ; c'est-à-dire par un graphe orienté contenant uniquement les arêtes de l'ordre qui ne se déduisent pas par transitivité.

La liste des fonctions à implanter, avec leurs documentations, leurs prototypes et quelques tests est disponible sur `http://www.lapcs.univ-lyon1.fr/~nthiery/RO/Notes/OrdresPartiels.mu`.

Vous pouvez trouver de l'inspiration dans le fichier suivant :`http://www.lapcs.univ-lyon1.fr/~nthiery/RO/Notes/exemples.mu`

Essayer d'évaluer la complexité au pire de chacune des fonctions que vous implantez.