Programmation Lineaire

- Numero: 1
- Prof: Siarry Patrick
- Date: 9 Octobre 2017

Programme

Rappels d'Algebre Lineaire

- Determinant
- Cofacteur
- Inversion de matrice
- Vecteur independant / lies
- Modelisation d'un probleme d'optimisation sous la forme d'un **programme lineaire (PL)**
- Resolution d'un PL a 2D -> resolution graphique
- Principe de l'algo du **SIMPLEXE**

Rappels d'Algebre Lineaire

On pourra trouver des exercices page 120 du livret de cours.

Determinant

Ordre 2

Example:

```
A = | 3 | -2 |
    | 4 |  5 |
```

`|A| = 3 * 5 - 4 * -2 = 23`

Factorisation:

```
| k |  k | = k * | 1 |  1 |
| 4 | 2k |       | 4 | 2k |
```

Un facteur commun dans une ligne (ou dans une colonne), on peut diviser toute la ligne / colonne par le facteur, mis a l'exterieur.

De meme:

```
| k |  k | = k * | k | 1 |
| 4 | 2k |       | 4 | 2 |
```

Ordre n (x > 2)

On developpe le determinent par rapport a une rangee (ligne ou colonne) pour se ramener a un determinent d'ordre `n - 1`.

Exemple:

```
A = | 2 |  0 | -1 |
    | 3 |  0 |  2 |
    | 4 | -3 |  7 |
```

On developpe / 1ere colonne.

```
+2 * |  0 | 2 |
     | -3 | 7 |
```

```
-3 * |  0 | -1 |
     | -3 |  7 |
```

```
+4 * | 0 | -1 |
     | 0 |  2 |
```

Il nous suffit de faire la somme pour trouver le determinent de `A`. On aurait pu etre plus intelligent en developpant la 2eme colonne. Car c'est la rangee qui comporte le plus de `0`.

```
-0 * | 3 | 2 |
     | 4 | 7 |
```

```
+0 * | 2 | -1 |
     | 4 |  7 |
```

```
-(-3) * | 2 | -1 |
        | 3 |  2 |
```

Il y a beaucoup moins de calcul a faire...

|A| = `3 * (2 * 2 - 3 * -1)` = `21`

Propriete pour faire apparaitre des zeros

One ne change pas la valeur du determinent si on remplace n'importe quelle ligne ou colonne par elle meme + une combinaison lineaire des autres lignes ou colonnes.

```
A = |  3 | 2 | -4 |
    |  1 | 0 | -2 | (On veut remplacer le  -2  par un  0 )
    | -2 | 3 |  3 |
```

On va faire `C3 += 2 * C1`

```
A = |  3 | 2 |  2 |
    |  1 | 0 |  0 |
    | -2 | 3 | -1 |
```

|A| = `-1 * (2 * -1 - 3 * 2)` = `8`

Cofacteur

Le cofacteur d'un element d'une matrice est le **sous-determinent** precede du *bon-signe*.

Example:

```
A = |  2 |  1 | -3 |  4 |
    |  5 | -4 |  7 | -2 |
    |  4 |  0 |  6 | -3 |
    |  3 | -2 |  5 |  2 |
```

Cofacteur a_{23} (la position du `7`):

```
  | 2 |  1 |  4 |
- | 4 |  0 | -3 |
  | 3 | -2 |  2 |
```

Remarque: `7` ne fait pas partie de son cofacteur.

La **matrice des cofacteurs** note `cof A`. Est obtenu en remplacant tous les elements de `A` par leurs cofacteur. (c'est un peu long a calculer)

```
cof A = | ? | ? | ? | ? |
        | ? | ? | ? | ? |
        | ? | ? | ? | ? |
        | ? | ? | ? | ? |
```

Inverse d'une matrice

C'est la matrice `A^-1^`:

$A * A^{-1} = A^{-1} * A = I$

`I` est la matrice identite soit:

```
A = | 1 | 0 | 0 |
    | 0 | 1 | 0 |
    | 0 | 0 | 1 |
```

Pour calculer la matrice inverse on va utiliser la matrice des cofacteurs.

A^{-1} = transposition (cof A) / det A La transposition est lorsqu'on echange les *colonnes* avec les *lignes*.

Exercice:

```
A = | 1 | 2 | 3 |
    | 2 | 3 | 4 |
    | 1 | 5 | 7 |
```

|A| = 2 != 0 donc A^{-1} **existe**

```
cof A = |  1 | -10 |  7 |
        |  1 |  4  | -3 |
        | -1 |  2  | -1 |
```

A^{-1} =

```
| 1/2 |  1/2 | -1/2 |
|  -5 |   2  |   1  |
| 7/2 | -3/2 | -1/2 |
```

Vecteurs independants / Vecteurs lies

`u1 = (1, 1, -1)` `v1 = (2, -3, 1)` `w1 = (8, -7, 1)`

(`u1`, `v1`, `w1`) forment-ils une **base** de *R3*? Ces vecteurs sont-ils independants?

```
det(u1 | v1 | w1) = |  1 |  2 |  8 |
                    |  1 | -3 | -7 |
                    | -1 |  1 |  1 |
```

Si det(`u1` | `v1` | `w1`) != 0 alors les vecteurs sont independants. Ici det(`u1` | `v1` | `w1`) = 0 donc: Les vecteus sont lies et ne forment pas une base. On peut ecrire n'importe lequel des 3 vecteurs comme une combinaison lineaire des 2 autres.

Programmation Lineaire (PL)

Tout probleme d'**optimisation** qui s'exprime *lineairement* en fonction des variables `xi` de decision.

Dans un probleme d'optimisation on a deux paties: * Fonction objectif `z` qui est a **minimiser** ou **maximiser**. * Contraintes a respecter concernant les **variables de decision** (les inconnues).

On dira qu'on a un PL si toutes les *contraintes* et la *fonction objectif* s'exprime **lineairement** en fonction des *variables de decision*.

Ces problemes sont tres frequents et on sait les resoudre grace a l'algo du **simplexe**

Exemple modelisation d'un probleme de PL (p. 121):

Les **variables de decision** sont donne: `X1`, `X2`, `HS`, `MP`, `PUB1`, `PUB2`

Contraintes: * main d'oeuvre: `0.75 * X1 + 0.5 * X2 <= 4 * 40 + HS` * temps machine: `1.5 * X1 + 0.8 * X2 <= 320` * matiere premiere: * `2 * X1 + X2 <= MP` * `MP <= 400` * `X1 <= 50 + 10 * PUB1` * `X2 <= 60 + 15 * PUB2` * `PUB1 + PUB2 <= 100`

Remarque: Les **variables de decision** sont positives ou nuls!

Fonction Objectif: `MAX z = benefice z = 15 * X1 + 8 * X2 - (6 * HS + 1.5 * MP + PUB1 + PUB2)Le salaire des employers est fixe donc pas variables il est inutile d'en prendre compte dans`z'.

Exemple 2 (Plus simple pour le resoudre) (p. 123):

Les **variables de decision** sont donne: `X1`: Fraction de tonne produit 1 `X2`: Fraction de tonne produit 2 `X3`: Fraction de tonne produit 3

Contraintes: * `12 * X1 + 52 * X2 + 42 * X3 >= 22` (Contraintes d'inegalite) * `2 * X1 + 2 * X2 + 10 * X3 >= 3.6` (contrainte d'inegalite) * `X1 + X2 + X3 = 1` (Contrainte d'egalite) * `X1, X2, X3 >= 0` (Regle d'or en PL)

Fonction Objectif: `MIN z = cout du produit` `z = 25 * X1 + 41 * X2 + 39 * X3`

Ceci est bien un probleme de PL. Car tout ca est bien lineaire. Grace a la contrainte d'egalite on va pouvoir reduire le nombre de variable. Et donc se ramener a un probleme en 2D.

On elimine `X1`. D'apres la contrainte num 3: `X1 = 1 - X2 - X3` **Nouvelles contraintes**: * `12 * X1 + 52 * X2 + 42 * X3 >= 22` donc `12 * (1 - X2 - X3)+ 52 * X2 + 42 * X3 >= 22` donc `4 * X2 + 3 * X3 >= 1`

- `2 * X1 + 2 * X2 + 10 * X3 >= 3.6` donc `2 * (1 - X2 - X3)+ 2 * X2 + 10 * X3 >= 3.6` donc `X3 >= 0.2`
- `X1 >= 0` donc `X2 + X3 <= 1`

Nouvelle Fonction Objectif: `MIN z = cout du produit` `z = 16 * X2 + 14 * X3 + 25 = 2 * (8 * X2 + 7 * X3)+ 25` `z' = 8 * X2 + 7 * X3` (On va chercher a minimiser `z'`)

Resumons:

```
{
  {4 * X2 + 3 * X3 >= 1
   X3 >= 0.2
   X2 + X3 <= 1}
  MIN: z' = 8 * X2 + 7 * X3
}
```

Resolution graphique: * On trace les droites de contraintes en conservant les bonnes partie du plan. `*` * On s'interesse a l'ensemble des points du plan pour lequel `z'` a une valeur donnee. `@` * On cherche le point # du plan (x2, x3) tel que: * On respecte les contraintes * On minimise `z'`

Info: La droite `z'` est une droite **ISO_COUT** Le coup lorsque l'on est sur la droite est le meme. On a pas une seule droite ISO_COUT mais une famille.

```
8 * X2 + 7 * X3 = C
X3 = -8/7 * X2 + C/7
```

```
    X3
    ^
   *|
    *
   @| *
    @  *
    |@   *
    | @   *
  * |   @   *
   *|    @   *
.3->*     @    *
    |*     @    *
    | *     @    *
```

```
 ******#******@***********
    |    *      @     *
    +----*------@----*----------------------------->X2
    0     ^*        @     *
         .25*
```

L'optimum # est donc:

```
x3 = 0.2
4 * x2 + 3 * x3 = 0.4
x2 = 0.1
x1 = 0.7

MIN z' = 1.8 + 1.4 = 2.2
MIN z = 2 * z' + 25 = 29.4
```

Generalisation

Si on generalise le probleme dans un espace de dimension `n`. * D est delimite par le **simplexe**. Dans notre exemple c'est un quadrilataire. * MIN `z'` revient a faire glisser un **hyper plan** et elle se fait sur un sommet du simplexe.

Il y a un cas particulier (**cas degenere**), il y a une infinite de solution le long d'une arrete du simplexe.

Principe tres general de l'algo du Simplexe

Rem: On ne peut pas calculer `z` en chacun des sommets et retenir le meilleur des sommets. Le nombre de sommet est tres grand et on ne connait pas facilement les coordonnees des differents sommet.

Voici les etapes de l'algo: 1. La recherche d'un sommet initial 1. On se deplace le long d'une arrete pour aller sur un sommet strictement meilleur 1. Check si on est sur l'optimum sinon reprendre l'etape precedente.

Grace a la linearite du probleme. On est sur de tendre vers une solution global.