

## **Tp Simplex**

Résoudre un programme linéaire grâce à la méthode du  
simplex - [git disponible ici](#)

Juliano Souza Luz, Thibault Capt

18.01.2023

## TP Simplex

### Choix du langage

Nous avons donc utilisé le Java afin de résoudre ce travail pratique. En effet, python est un langage qui est très lent à l'exécution, car celui-ci est un langage **interprété**. Contrairement au Java, qui est un langage compilé. Nous pouvons ainsi essayer de gagner du temps d'exécution pour la résolution du problème.

### Astuces utilisées

Ensuite, dans la plupart des cas, nous avons utilisé des **array** à 1 ou 2 dimension(s) qui est bien plus rapide que les Listes Java. Lien disponible [ici](#).

Nous avons aussi opté pour que l'utilisateur puisse choisir un mode débogage ou non, ce qui permet de supprimer presque tous les prints en mode non-débogage et gagner énormément de temps d'exécution.

Nous commençons également à calculer le temps d'exécution après le retour du parseur, soit au début des appels de fonctions du simplex, et nous terminons le calcul avant l'affichage des résultats. Nous faisons ce choix, car l'affichage de matrice est assez gourmand en temps et cela se remarque de plus en plus en fonction de la taille de la matrice et du fichier original.

### Analyse des performances

Nous avons reçu plusieurs fichiers à tester pour vérifier le bon fonctionnement de notre code.

Temps d'exécution et pivots : Méthode 2 / Méthode 1 (Les méthodes sont expliquées un peu plus bas)

- lp\_glaces.txt
  - Valeur obj : -34.25
  - nombre de pivots : 3 / 2
  - Temps d'exécution : 1ms / 0ms
- lp\_test\_non\_admissible.txt
  - Valeur obj : -34.25
  - nombre de pivots : 5 / 8
  - Temps d'exécution : 1ms / 1ms
- lp\_test.txt

- Valeur obj : 4.5000000000000015
  - nombre de pivots : 11 / 16
  - Temps d'exécution : 1ms / 0ms
- lp\_test2.txt
  - Valeur obj : ~
  - nombre de pivots : ~ / ~
  - Temps d'exécution : ~ / ~
  - Ce fichier ne nous retourne pas de résultat.
- network1.txt
  - Valeur obj : 78768.0
  - nombre de pivots : 22 / 26
  - Temps d'exécution : 4ms / 5ms
- network2.txt
  - Valeur obj : -0.0
  - nombre de pivots : 31 / 34
  - Temps d'exécution : 17ms / 18ms
- network3.txt
  - Valeur obj : 84077.0
  - nombre de pivots : 132 / 183
  - Temps d'exécution : 45ms / 46ms
- network4.txt
  - Valeur obj : 188508.0
  - Nombre de pivots : 1049 / 2628
  - Temps d'exécution : 227ms / 369ms
- network5.txt
  - Valeur obj : 123542.0
  - Nombre de pivots: 4074 / ~
  - Temps d'exécution : 9114ms / ~

Le temps d'exécution est, dans l'ensemble, assez faible, bien qu'il soit encore possible de l'améliorer.

Une grande amélioration du temps que nous avons pu constater est la manière de choisir le négatif lors du choix du pivot. En effet, nous pouvons choisir cela de deux façons différentes.

En premier lieu, nous pouvons choisir le 1er négatif rencontré. Ce qui marche très bien pour les problèmes ne comportant pas une trop grande taille. Cependant, avec cette méthode, le network5 nous retourne une boucle infinie.

En second lieu, nous pouvons choisir de comparer les négatifs obtenus et choisir le pivot sur la ligne du plus petit négatif rencontré. De cette manière, tout d'abord le network5 nous retourne un résultat, mais en plus les autres inputs voient une diminution du temps d'exécution. C'est donc pour cela que nous choisissons donc cette méthode par défaut et notre jar généré utilise donc cette 2<sup>e</sup> méthode.

## **Problèmes rencontrés**

Nous avons rencontré plusieurs problèmes au cours de ce travail pratique.

Premièrement, lors de la phase 1, nous ne gérons pas tous les négatifs des membres de droite. En effet, nous passons en paramètre la ligne où nous trouvons un négatif et remplaçons la ligne en question. Le problème étant que s'il y avait plusieurs négatifs dans les membres de droite, ils y resteraient.

Deuxièmement, nous avons utilisé pendant la quasi-intégralité du temps imparti la première méthode de sélection du pivot (négatif décrit au-dessus). Nous n'avions donc aucun résultat pour network5 ainsi que pour certains autres fichiers de tests que nous testions. En changeant cette méthode par la seconde, nous avons ainsi maintenant un résultat, qui nous l'espérons, est correct.

Troisièmement, en jouant un peu avec notre epsilon, nous avons constaté que nous pouvions parfois obtenir des résultats différents. Par exemple, le fichier lp\_test.txt nous retourne le bon résultat quand notre epsilon vaut  $1e-7$  (qui est notre valeur par défaut que nous utilisons depuis le début et que nous avons choisie arbitrairement). Si nous dépassons  $1e14$ , nous remarquons que nous n'obtenons plus de résultat pour ce fichier. Il s'agit sûrement de la valeur de la solution optimale en phase 1 qui n'est plus considérée à partir de cette valeur. Il pourrait aussi s'agir d'un dépassement de floating après la virgule, mais nous n'en sommes vraiment pas sûrs.